Universidad Zaragoza
1542

Trabajo Fin de Grado

Desarrollo de herramientas gráficas para facilitar la creación de juegos pervasivos en espacios interactivos

Autor

Alejandro Navarro Castillo

Directores

Dr. Javier Marco Rubio

Dra. Eva Cerezo Bagdasari

Escuela de Ingeniería y Arquitectura

2018

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

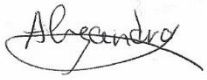
D./Dª. Alejandro Navarro Castillo,

con nº de DNI 2502274R en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Desarrollo de herramientas gráficas para facilitar la creación de juegos pervasivos en espacios interactivos,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, a 29 de Agosto de 2018



Fdo: Alejandro Navarro Castillo

Desarrollo de herramientas gráficas para facilitar la creación de juegos pervasivos en espacios interactivos

RESUMEN

El grupo de investigación de la Universidad de Zaragoza, el AffectiveLab, lleva varios años investigando y desarrollando para el campo de la computación ubicua, en especial para el campo de los juegos pervasivos, que son juegos en los que la interacción con objetos en el plano real conlleva reacciones en el plano digital.

El desarrollo de este trabajo fin de grado se encuentra dentro de un proyecto de investigación a nivel nacional llamado JUGUEMOS, cuya finalidad es la creación de juegos pervasivos, para lo cual se ha desarrollado un toolkit conocido también como JUGUEMOS para facilitar todo el proceso de desarrollo. De forma más específica, el trabajo corresponde con un hito dentro del proyecto, que es la creación de interfaces gráficas para facilitar su implementación.

El toolkit JUGUEMOS permite, utilizando un lenguaje específico conocido como TUIML, definir los componentes que participarán en el juego, así como las distintas manipulaciones que se puedan realizar sobre ellas.

El objetivo de este proyecto es desarrollar una interfaz gráfica que se integre con el toolkit JUGUEMOS, y facilite a diseñadores y desarrolladores de juegos pervasivos la implementación de ciertas partes de código, como la definición de reglas TUIML que determinan la interacción del juego pervasivo, siendo los desarrolladores capaces de modificar de forma gráfica y en tiempo real dichas reglas TUIML utilizando una interfaz gráfica.

Para ello, se siguieron diversos pasos hasta alcanzar el producto final:

- Se llevó a cabo un análisis del toolkit: cómo funciona, cuáles son sus tareas, cómo habría que implementar la aplicación.
- Se llevaron a cabo varias iteraciones sobre el diseño de la interfaz, para mejorar cuestiones de usabilidad, y se diseñó la arquitectura software general del sistema.
- Paralelamente con el diseño de la aplicación, se procedió a su implementación e integración con el espacio interactivo.
- Para comprobar que la aplicación funcionaba correctamente, se realizaron una serie de pruebas, tanto funcionales como pruebas con usuarios para comprobar cuestiones de usabilidad.

Tabla de contenido

1 Contexto y objetivo del trabajo

1.1 Contexto de desarrollo del trabajo

Este trabajo de fin de grado ha sido desarrollado en el seno del grupo de investigación AffectiveLab [GIG14], especializado en el área de Interacción Persona-Ordenador. El trabajo de este grupo se ha especializado en cuatro áreas principales: el desarrollo de humanos virtuales para ser usados como interfaces multimodales en aplicaciones en tiempo real, el desarrollo de interfaces de usuario tangibles, la consideración de los aspectos afectivos en la interacción con el usuario, y la creación de interfaces accesibles y usables para usuarios con necesidades especiales.

El grupo de investigación lidera un proyecto de investigación a nivel nacional, en el que participan otros dos grupos de investigación de la Universitat de les Illes Balears y de la Universidad de Granada, conocido dicho proyecto como JUGUEMOS (TIN-2015-67149-C3-1R). El proyecto tiene como objetivo el desarrollo de juegos pervasivos orientados a niños en los que elementos físicos se entremezclan con el mundo digital de manera que el usuario se sirve de acciones sobre dichos elementos físicos para interaccionar con lo digital.

1.2 El espacio interactivo

En el marco del proyecto JUGUEMOS y con la finalidad de explorar las nuevas posibilidades de los juegos orientados a niños basados en interacción física, el grupo ha creado un espacio, que se encuentra en el laboratorio cesAr en el edificio Etopia, una infraestructura dependiente del Ayuntamiento de Zaragoza dirigida al desarrollo de nuevas tecnologías, y reservada para los proyectos más creativos e innovadores en Aragón (ver Figura 1).

Imagen 1: Espacio interactivo en el edificio Etopia

El espacio interactivo cuenta con diversos elementos hardware que recogen los datos del entorno y reaccionan según la interacción del usuario con el espacio:

- Tabletops tangibles: El usuario interacciona mediante la manipulación de objetos físicos sobre la superficie de la mesa (interacción tangible). Para que la mesa pueda reconocer y diferenciar objetos, cada objeto tiene un código identificativo(*fiducial*) en

forma de dibujo pegado en la base del objeto (ver imagen 2) y el sistema hace uso de un software de reconocimiento (Reactivision [REA]).

Imagen 2: Objetos con fiduciales

- Real Time Localization System (RTLS): El entorno dispone de un sensor RTLS de la marca Ubisense, que se encarga de capturar la localización de objetos y usuarios del espacio en tiempo real. El sistema está formado por 4 sensores de posición que rastrean los dispositivos programados para ello (llamados *tags*), devolviendo sus coordenadas en el espacio.
- Kinects: Kinect es un periférico desarrollado por Microsoft que incluye una cámara que detecta y posiciona las articulaciones del usuario: pies, rodillas, cadera, hombros, codos… de forma que es posible reconstruir en tiempo real la pose de los jugadores.
- Proyectores: Los proyectores están colocados en el techo y permiten proyectar imágenes o vídeos en las paredes del espacio para añadir inmersión a los juegos.
- Altavoces: El sistema también incluye altavoces, tanto en las mesas, como externos, que emiten sonidos de distintos tipos para acompañar al juego y que la experiencia sea más inmersiva.
- Foco RBG: El foco permite emitir luz con intensidad y color regulables.
- Actuadores: Los actuadores son los dispositivos que comunican información a los jugadores, como puede ser un dispositivo Android, o la superficie de las paredes del espacio.

La red de componentes no es una red cerrada, ha sido diseñada con la flexibilidad, y con la idea de que se pueda ampliar con otros dispositivos que expandan las posibilidades de creación de juegos pervasivos, ampliando las posibilidades de detección o intervención sobre el espacio físico del espacio interactivo. Este tipo de espacios interactivos se enmarcan en el paradigma de la *computación ubicua*, en la que la capacidad informática y la interacción se diluyen en el entorno. Dicha integración de las tecnologías ubicuas, así como el desarrollo de juegos pervasivos para este espacio es posible gracias al toolkit JUGUEMOS.

1.3 El toolkit JUGUEMOS

Dentro del proyecto JUGUEMOS una de las tareas fundamentales ha sido el diseño y desarrollo de un toolkit software que facilite el desarrollo y codificación de juegos pervasivos para el espacio descrito anteriormente. El toolkit JUGUEMOS permite tanto la integración de distintas tecnologías ubicuas (sensores y actuadores) como el desarrollo de la lógica del juego (la evolución del juego en base a las manipulaciones de los jugadores)

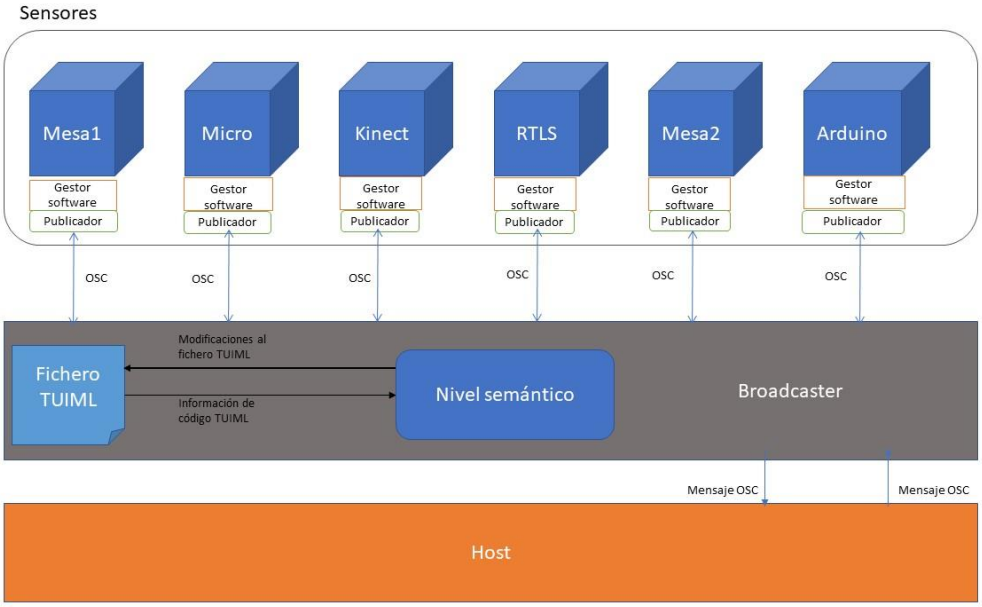


Imagen 3: Diagrama de componentes del sistema

En la imagen 3 se muestra la arquitectura del espacio interactivo, formado por tres bloques de dispositivos separados que se comunican entre sí:

Los **sensores**, que son dispositivos conectados al espacio interactivo que recogen valores del entorno físico; la posición de algún objeto, los sonidos recogidos por un micrófono, o los movimientos de la mano de algún jugador recogidos por las cámaras. Cada dispositivo tiene implementado un gestor software, que tiene como finalidad la recogida y captura de estos valores del entorno físico. Además, cada gestor proporciona un proceso publicador, cuya tarea es la gestión del proceso de comunicación con los otros elementos del espacio interactivo.

Los **actuadores**, que son dispositivos que actúan sobre el entorno interactivo modificando propiedades físicas de dicho entorno: sonido, movimientos de objetos del entorno…Es el Host el que ordena el momento en que los actuadores actúan, y el broadcaster el que se encarga de redirigir la orden al proceso publicados asociado al actuador destino

El **Broadcaster** (o gestor de publicadores), mantiene las conexiones con los sensores, y recibe los mensajes enviados por ellos, de manera individual, gracias a la IP de cada sensor que lo identifica. El Broadcaster contiene un componente software, conocido como **nivel semántico**, que se encarga de filtrar los valores de los paquetes recibidos, utilizando unas reglas que permiten filtrar, o relacionar ciertos valores y/o interacciones con la lógica del juego a desarrollar. El broadcaster obtiene las reglas sobre las cuales se basa la interacción del juego pervasivo leyendo las mismas de un fichero con código en formato TUIML (*Tangible User Interface Modeling Languaje*) [SJ09].

El **Host**, que es el componente que contiene la lógica del juego pervasivo, encargado de la lógica del juego, y de reaccionar según la interacción física del usuario.

Las bases del sistema son el protocolo que utiliza el espacio interactivo para la comunicación entre equipos, y cuál es el lenguaje que se utiliza para definir las reglas de los juegos.

1.3.1 Protcolo de comunicación

Informáticamente el espacio interactivo se configura como un sistema distribuido, en el que los distintos componentes previamente descritos se interconectan entre sí mediante una red local (Ethernet y wifi). Los sensores y el broadcaster del sistema interactivo se comunican mediante el paso de mensajes del formato *OSC* (Open Sound Control) [W05], un protocolo de comunicación en red basado en sockets UDP, diseñado para las interconexiones de dispositivos digitales con características muy distintas entre sí. Se ha elegido el protocolo OSC ya que permite realizar envíos por broadcast así como envíos masivos de mensajes, en comparación con otros protocolos.

Los mensajes en OSC están divididos en paquetes, compuestos por:

-Un campo *address* que representa la dirección del nodo a conectarse.

-Un número indefinido de parámetros de tipo base: integer,char,float…

1.3.2 Lenguaje TUIML

Para la definición de las reglas de interacción física en el contexto de un juego pervasivo, el proyecto JUGUEMOS ha decidido utilizar el lenguaje **TUIML** [SJ09], que es un conjunto de términos y sintaxis que permite expresar una aplicación basada en interacción física. Además, este lenguaje puede ser expresado en un lenguaje entendible por ordenador, por ejemplo XML (*Extensible Markup Language*).

Los conceptos del TUIML se dividen en:

- Los *tokens*, que son objetos físicos que representan información digital. Un objeto físico se considera un token cuando está ligado a una variable. Un usuario interactúa con un token para acceder o manipular la información digital asignada al mismo.
- Los *Constraints*, que son restricciones físicas que limitan el comportamiento de un token con el que está asociado, respecto a la manipulación de dicho token sobre el constraint.

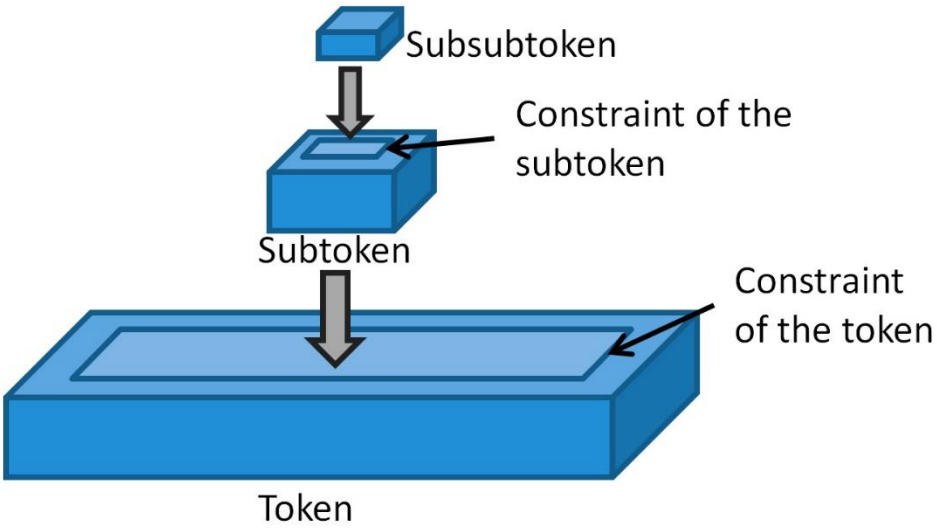


Imagen 4: Diagrama que representa los distintos tipos de tokens

En la imagen 4, los prismas de color azul más oscuro representan los tokens, subtokens y subsubtokens, y las áreas de color azul más claro reflejan los distintos constraints.

1.4 Objetivo del trabajo

El objetivo de este trabajo fin de grado es llevar a cabo el análisis, diseño e implementación de una interfaz gráfica de usuario (GUI) que facilite la labor del desarrollador de juegos pervasivos con el toolkit JUGUEMOS.

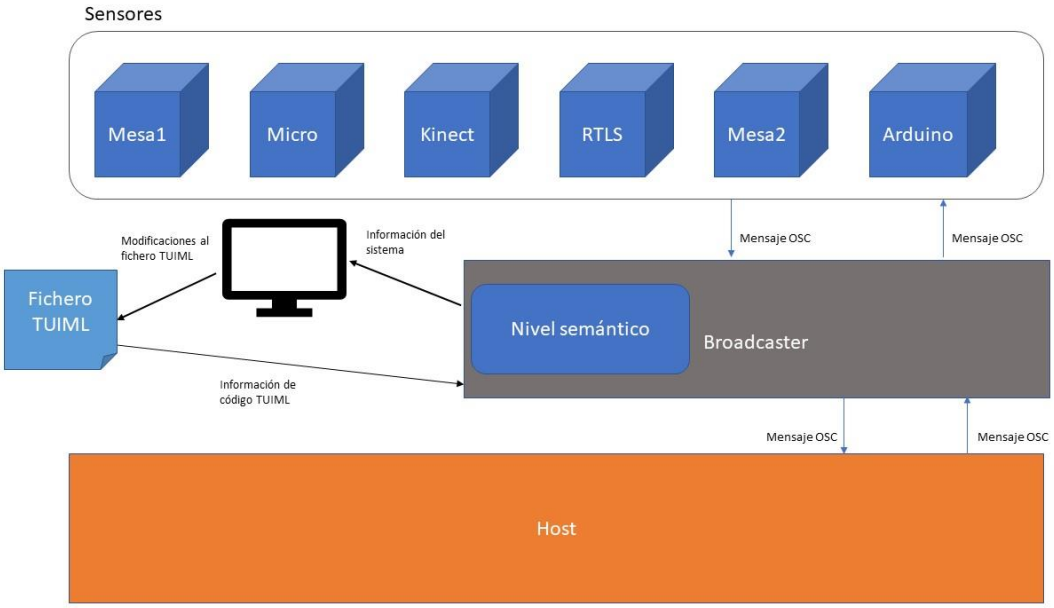


Imagen 5: Diagrama de componentes del sistema tras el desarrollo del proyecto

La interfaz gráfica se integrará en la arquitectura del toolkit (Imagen 5) de forma que recibirá la información recogida por el nivel semántico, se la presentará al desarrollador en forma gráfica, y recogerá cualquier modificación de esta para poder alterar el modelado TUIML.

Gracias a la interfaz se facilitará y ayudará a la creación de juegos pervasivos en esta plataforma, permitiendo que diseñadores y desarrolladores no expertos desarrollar juegos de

manera sencilla, facilitándoles la creación de la lógica, logrando así hacer más pequeña la brecha que hay entre el proceso de diseño de un juego pervasivo y su codificación.

1.5 Estructura de la memoria

A continuación, se van a explicar de forma breve las diversas secciones de la memoria.

1. En el **Contexto y objetivo del trabajo** se realiza una introducción sobre el estado de los juegos pervasivos y de la computación ubicua dentro de la Universidad de Zaragoza, explicando cuál es el estado actual de las tecnologías y espacios disponibles para el desarrollo de los juegos, además de introducir los objetivos de este proyecto.
2. En el **Estado del arte** se resume de forma breve la situación actual de la computación ubicua, las aportaciones del grupo AffectiveLab, y cómo impactará este proyecto el ámbito de computación ubicua.
3. En el **Análisis del problema a resolver** se presenta un análisis del toolkit JUGUEMOS para obtener los requisitos que definan la funcionalidad del proyecto.
4. En el **Diseño del sistema** se discute cuáles han sido las decisiones de diseño tomadas para la interfaz de la aplicación y su arquitectura software.
5. En la **Implementación** se discute cómo han sido las iteraciones de implementación de la interfaz, así como una explicación de las clases implementadas, y diversas cuestiones y problemas relacionados con la codificación.
6. En las **Pruebas del sistema** se muestran las pruebas a las que fue sometida la aplicación para garantizar su correcto funcionamiento.
7. Y, por último, en el apartado de **Conclusiones y trabajo futuro** se realiza una recapitulación del desarrollo del trabajo, y se termina con objetivos del trabajo futuro.

Además, la información relacionada con la documentación del proyecto se incluye en los anexos, al final de la memoria:

- **Anexo A**: Aquí están incluidos los diagramas de secuencia de la aplicación, mostrando la interacción entre las diversas partes del sistema.
- **Anexo B:** Aquí se incluyen los casos de usos del sistema especificados, con información respectiva al autor, precondiciones y postcondiciones, y flujo de eventos de cada caso de uso.
- **Anexo C:** En este anexo se incluye un ejemplo de fichero TUIML, describiendo los campos del fichero y las relaciones entre ellos.
- **Anexo D:** En este anexo se incluye planificación temporal y de tareas del proyecto.
- **Anexo E:** En este anexo se procede a explicar de forma más extensa los atributos y métodos de las clases del diagrama de clases.

2 Estado del arte

El entorno interactivo se desarrolla dentro del área de la *computación ubicua* [W93], concepto que representa la integración del mundo físico con el mundo virtual para interactuar con el sistema. Una de las características de la *computación ubicua* es lograr que el usuario interactúe con el sistema de manera indirecta, difuminando la línea entre ambos mundos.

Dentro del área de la *computación ubicua*, es destacable la idea de *juegos pervasivos* [MCMN05], término que se refiere a todos aquellos juegos donde la experiencia de juego se extiende hasta el mundo real, o aquellos donde el mundo virtual es mezclado con el mundo físico.

La finalidad del proyecto JUGUEMOS es avanzar en la creación de juegos pervasivos y en introducir facilidades para que todos los desarrolladores no expertos en el espacio puedan diseñar dichos juegos. En la publicación de Steve Hinske [HLMR07], se defiende que "*los juegos pervasivos extienden la experiencia de juego hacia el mundo real*", mientras que "*el jugador se desprende de las cadenas de la consola y experiencia un juego que está interconecta con el mundo real y está disponible en cualquier lugar y en cualquier momento*". El término fue introducido por IBM en 1998 como un paradigma que trata con la integración de ordenadores en nuestro entorno.

La computación ubicua se encuentra estrechamente relacionada con el campo de desarrollo de interfaces tangibles (TUI), un tipo de interfaces que pretende trabajar con software de realidad mixta. El concepto de TUI fue propuesto en 1997 por Ishii y Ulmer [IU97], en un artículo donde aspiraban a darle una profundidad extra al concepto de interfaz gráfica, aferrada a sistemas "clásicos" informáticos, con elementos rectangulares con los que se interactuaba mediante ratón y teclado.

El grupo AffectiveLab lleva muchos años investigando sobre este campo de computación ubicua e interfaces tangibles. Una de sus aproximaciones a dicho campo se recoge en [MCB08], donde se habla de *NikVision*, un prototipo de tabletop orientada a videojuegos para niños. Dicho prototipo funciona recogiendo la posición y orientación de elementos tangibles usando una cámara con software de reconocimiento y análisis de imagen. La imagen recogida por la cámara se representará virtualmente en una pantalla conectada a *NikVision*.

Imagen 6: Pototipo NikVision

El grupo AffectiveLab ha desarrollado varios juegos usando el lenguaje TUIML para dispositivos con interfaces tangibles [MCB12]. Por ejemplo, en [CMB15] se explica el proceso de desarrollo de un juego de granjeros para el tabletop NIKVision. Dicho juego utiliza una TUI para su interacción con el usuario: los usuarios interactúan con el sistema mediante la manipulación de unos juguetes físicos que representan animales, sobre la superficie del tabletop. En este caso específico, los animales actúan como tokens, que pueden ser puestos, movidos y generados en el tabletop, generando distintas salidas en el sistema. Cada *token* tiene asociado un identificador que le hace único del resto. La mesa también actúa como un *token* el cual permite la adición de otros tokens. La mesa tiene también ciertas áreas físicas que son las que actúan como constraints, limitando la interacción, o generando nuevos eventos al añadir o quitar ciertos tokens a los constraints.

Para facilitar la definición del TUIML es estos juegos, el grupo AffectiveLab diseñó el toolkit ToyVision [MCB12] para, de forma gráfica, definir el TUIML asociado a dichos juegos. El toolkit ToyVision es "*un toolkit para prototipar juegos tangibles en aparatos tabletop basados en la visión"*, donde al usuario se le permite poder añadir distintos tokens a un juego pervasivo para una mesa interactiva con un asistente gráfico.

Incluye un editor [GLC15] que permite definir la estructura de los juegos pervasivos en TUIML, para facilitar su desarrollo. El editor muestra gráficamente la ubicación de los objetos tangibles sobre el tabletop, y permite definir las manipulaciones asociadas a dichos objetos.

Este TFG expande la idea del editor gráfico descrito anteriormente, ya que la finalidad es facilitar la creación de juegos pervasivos en espacios interactivos, dando soporte a distintos tipos de manipulaciones físicas, y que, a diferencia del proyecto fin de carrera anterior, pueda soportar los distintos tipos de sensores y tecnologías ubicuas del espacio interactivo, y no sólo soportar el prototipo NikVision.

3 Análisis del problema a resolver

Como se ha visto anteriormente, el objetivo del trabajo es minimizar la brecha entre diseñadores y desarrolladores facilitando la generación rápida de prototipos de juegos pervasivos para el espacio JUGUEMOS. En particular, con la GUI que se pretende desarrollar se quiere acelerar el proceso de codificación de un juego pervasivo y su lógica y ayudar a los diseñadores a poder entender y generar parte del código de una manera visual.

Una vez se tiene una visión general sobre el estado del arte y el estado del trabajo, y los objetivos y alcance de la aplicación, se va a proceder a efectuar un análisis de cómo debe ser el software para cumplir estos requisitos. Como primera decisión de análisis, se ha decidido estudiar cómo funciona el sistema del toolkit Juguemos, y cuáles son las funcionalidades necesarias para este proyecto. Tras este periodo de análisis del toolkit JUGUEMOS, se han empleado técnicas de análisis de software recogidas en la metodología relacionada con la Ingeniería de Requisitos: el análisis de requisitos funcionales y la recolección de casos de usos.

3.1 Análisis de la arquitectura del espacio interactivo JUGUEMOS

Para conocer cuáles son las necesidades de este trabajo, se ha de entender, a un nivel más profundo, cómo ha sido el diseño e implementación de la arquitectura del toolkit JUGUEMOS, descrita en la imagen 3, y cómo funcionan los servicios de los que depende el toolkit, en concreto, la estructura de los mensajes OSC, y la forma de implementación del TUIML:

3.1.1 Estructura del mensaje OSC en el toolkit JUGUEMOS

En el toolkit JUGUEMOS, la comunicación mediante el protocolo OSC se divide en dos fases:

En la primera fase el sensor en cuestión debe conectarse con el Broadcaster con un mensaje de inicio de conexión (ver Tabla 1):

Address (string)	Parámetro(int)	Nombre (String)	Tipo (String)
/host/connect	puerto	nombre	0D|1D|2D
/sensor/connect	puerto	nombre	0D|1D|2D
/display/connect	puerto	nombre	0D|1D|2D

Tabla 1: Campos de un mensaje OSC de conexión

En este mensaje hay cuatro campos:

- El campo *Address*: donde se indica la intención de iniciar una conexión con el broadcast y el tipo de proceso del que se trata, habiendo 3 tipos distintos:
    1. *Host*: Proceso con el código del juego pervasivo
    2. *Display:* Proceso asociado a una pantalla donde se va a mostrar información del espacio interactivo.
    3. *Sensor:* Publicador asociado a un gestor a integrar en el espacio interactivo.

- El campo *puerto*: Un entero que especifica el puerto del que dicho proceso escuchará mensajes del broadcast.
- El campo *nombre*: Que indica el nombre del proceso.
- El campo *tipo*: Se refiere al tipo de proceso conectado. El tipo puede ser de tres formas: 0D,1D, o 2D, y cada uno de ellos se explicará con más detalle en la sección 3.1.2.

Una vez conectados, los gestores mandan la información captada por los sensores en un formato estándar, que se muestra en la Tabla 2.

Address(string)	Manipulación (string)	ID objeto (int)	ID sesión (int)	Valor(es) (float)
/sensor/0D/*nombre*	Add			0|1
/sensor/1D/*nombre*	move/rotate			0-1
/sensor/2D/*nombre*	move			Coordenadas (x,y)

Tabla 2: Formato estándar de los mensajes OSC tras la conexión

El campo *Address* es el primer campo del protocolo OSC e incluye el tipo de mensaje del protocolo, y el tipo de datos de los valores recogidos. En este caso el emisor del mensaje será un sensor con un nombre determinado, y el tipo de los valores del mensaje puede ser 0D,1D o 2D.

Otro de los campos del mensaje es el campo *manipulación*, que sirve para que los sensores puedan enviar datos numéricos de las manipulaciones de los usuarios en el espacio interactivo. Estas manipulaciones pueden ser: add (poner o quitar un objeto), move(mover un elemento físico en el espacio), y rotate(cambiar la orientación de un objeto físico en el espacio).

El campo *ID objeto* representa el nombre identificativo del objeto.

El campo *ID sesión* representa las instancias de objetos con el mismo *ID objeto.* Por ejemplo, si tenemos dos instancias del objeto con *ID objeto* "Regadera", una tendrá el *ID sesión* 0 y la otra el *ID sesión* 1.

El campo *Valor* representa los datos recogidos por el sensor, y dependen del tipo de sensor que sean, 0D, 1D o 2D. En el siguiente punto se explicará con más detalle la clasificación de los sensores en 0D,1D y 2D.

3.1.2 Implementación del lenguaje TUIML en el toolkit JUGUEMOS

Como se puede observar en la imagen 3 y se explicó en el apartado 1.4.2, los elementos del lenguaje TUIML son los *Tokens,Subtokens* y los *Constraint.*

Dependiendo de los grados de libertad espacial que el constraint impone al subtoken, un constraint se puede clasificar en 0D,1D,2D y 3D, aunque sólo los tres primeros están implementados en el toolkit JUGUEMOS.

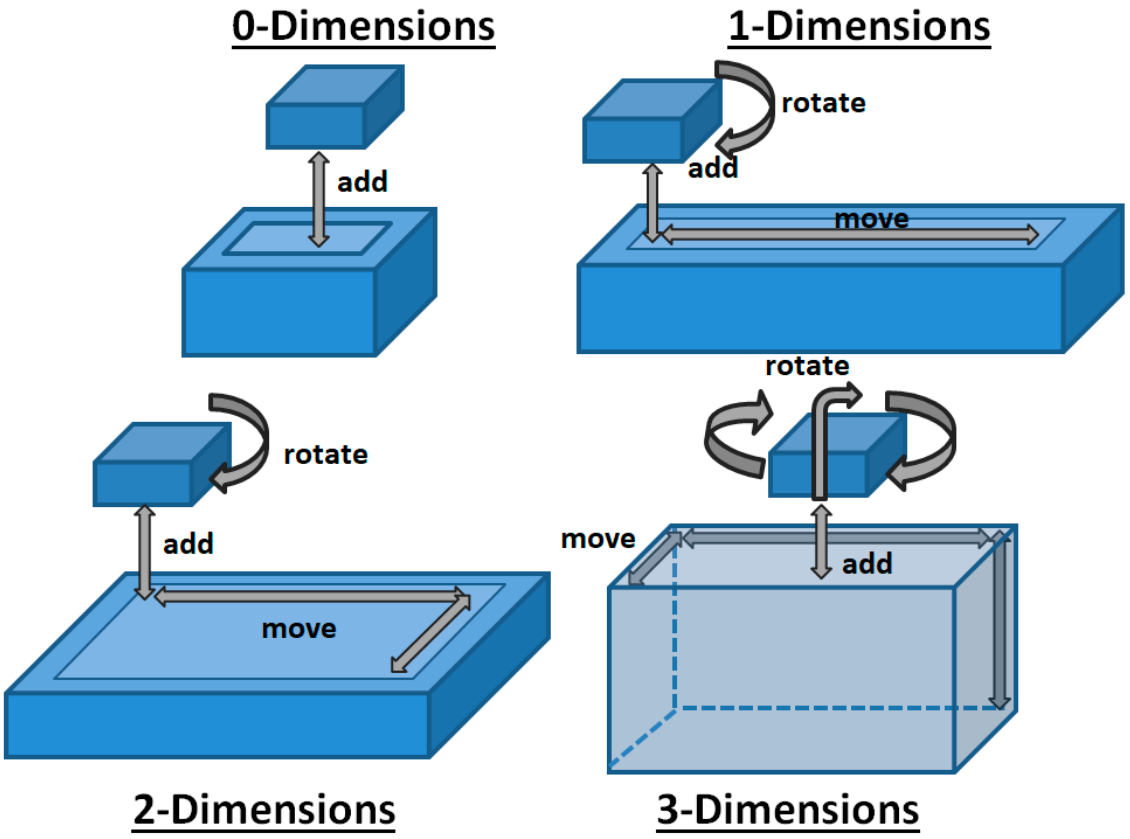


Imagen 7: Distintos tipos de Tokens y subtokens

Como se escenifica en la imagen 7, los constraints tienen diversos formatos de representación:

- 0D: Un valor binario, que toma entre los valores verdadero y falso. La única forma en la que se puede manipular un *Subtoken* dentro de un *Constraint* 0D es añadiéndolo(*add*) a un constraint. Por ejemplo: un botón de un dispositivo. Dicho dispositivo con botón se compone del *token* (dispositivo), *subtoken* (botón) y el *constraint*(mecanismo que restringe el movimiento del botón a pulsado/despulsado)
- 1D: Un valor unidimensional, que oscila entre el rango 0-1, con una precisión de dos decimales. Un subtoken dentro de un constraint 1D puede ser manipulado mediante la adición, movimiento o rotación de un subtoken en un constraint. Por ejemplo, un ábaco. El marco del ábaco (token), impone una restricción (constraint) a los subtokens (bolas) de moverse en línea (1D).
- 2D: Un valor bidimensional, que representa un vector de pares de coordenadas (x,y). Un subtoken dentro de un constraint 2D puede ser manipulado también mediante la adición, movimiento o rotación de un subtoken en un constraint, dichas acciones descritas con dos grados de libertad (horizontal y vertical).Un ejemplo son los objetos(subtokens), que se mueven en la superficie(constraint) de una mesa (token)
- 3D: Un valor tridimensional, que representa una tupla de 3 coordenadas en el espacio (x,y,z). Un token 3D puede ser manipulado también mediante la adición, movimiento o rotación de un subtoken en un constraint.
  Un ejemplo son los objetos (subtokens) que se mueven en la superficie (constraint) de una habitación (token).

El estado concreto de un sistema interactivo en un momento determinado se representa mediante una lista de TACs (Token&Constraints). Un TAC representa el estado en un momento determinado de un subtoken respecto a un token. Por ejemplo, si tenemos un *token* como puede ser un tabletop, el cual tiene ciertas restricciones físicas (su superficie activa), el estado de los objetos que están siendo manipulados sobre su superficie se representa mediante una lista de TACs, que informarán de las distintas posiciones y orientaciones de los objetos colocados sobre la superficie activa de la mesa. Además, una manipulación de un *subtoken* dentro de estas áreas restringidas de nuestro *token* es lo que genera diferentes tipos de comportamientos e interacciones.

En el toolkit JUGUEMOS, la relación entre *tokens* y *constraints* queda definida usando el lenguaje TUIML, con el que cada desarrollador definirá las reglas deseadas para el funcionamiento de su juego y con el que se expresarán las manipulaciones que se están realizando en el espacio interactivo mediante una lista de TACs que se va actualizando conforme el juego pervasivo evoluciona. Usando este lenguaje TUIML, el proceso Broadcaster (Véase la Imagen 4) es el que se encarga de mantener y crear dicha lista de TACs, además de filtrar los mensajes recibidos de acuerdo con dicha lista de TACs. Para ello, debe conocer los diferentes tokens, constraints y subtokens que participan en el juego pervasivo. Con ese fin, hace uso de la información especificada en un fichero XML, en cual se definirán todos los elementos TUIML. Por último, el Broadcast se conectará con el código del juego pervasivo, mediante una API, la cual informará al juego de los valores de los elementos. La API se conecta con el Broadcast, también mediante mensajes OSC. La API y el broadcaster se han mantenido separados para permitir que el desarrollo de juegos sea posible hacerlo en cualquier lenguaje de programación, siempre que permita la comunicación mediante mensajes OSC.

Estructura de un fichero TUIML:

```
<jueguemos>
        <token name="Nombre del Token">
                <constraint name="Nombre del constraint" type= "0D|1D|2D" list_vertex="Lista de valores del constraint(0.0,0.1…n)">
                        <tac subtoken="Nombre del subtoken"/>
                </constraint>
        <subtoken idG="Nombre del subtoken"/>
                <constraint name="Nombre del constraint" type= "0D|1D|2D" list_vertex="Lista de valores del constraint(0.0,0.1…n)">
                        <tac subsubtoken="Nombre del subsubtoken"/>
                </constraint>
                        <subsubtoken idG="Nombre del subsubtoken"/>
        </subtoken>
        </token>
</jueguemos>
```

Imagen 8: Ejemplo básico de un fichero TUIML

La imagen 8 muestra la estructura general que tiene un fichero TUIML:

- Las etiquetas *token* incluyen el nombre del mismo, y pueden contener varias etiquetas *constraint* además de las etiquetas *subtoken* y *subsubtoken*.
- Las etiquetas *constraint* incluyen como atributos su nombre, el tipo de constraint que es, ya que puede ser 0D,1D o 2D, y la lista de valores que determinan la restricción, los cuales cambiarán según al tipo de constraint que representen.
- Las etiquetas *tac* incluyen una lista de todos los *subtokens* a los cuales se le aplicarán a dicho *constraint*. Si algún *subtoken* se introduce en el área determinada como *constraint*, el Broadcast recibirá la notificación del semántico, puesto que éste habrá filtrado dicha información.
- Las etiquetas *subtoken* y *subsubtoken* mantienen una estructura similar a la etiqueta *token*, pero sólo el *subtoken* puede tener constraints.

Se puede observar un ejemplo concreto comentado en el Anexo B.

3.2 Requisitos funcionales y no funcionales

Tras estudiar el envío y recibo de mensajes OSC, además de las reglas usadas para limitar la interacción, se tiene una idea mucho más clara de cómo debe ser el trabajo de fin de grado, y de qué funcionalidades le debe proporcionar al usuario.

Para ello se ha elaborado un proceso para obtener y recoger los requisitos funcionales y no funcionales de la aplicación al inicio de la vida del desarrollo del producto.

RF-1	El sistema permitirá al usuario visualizar los datos recibidos de los sensores 0D,1D y 2D.
RF-2	El sistema muestra una lista de sensores conectados
RF-3	El usuario podrá exportar el TUIML a un fichero .xml
RF-4	El usuario podrá marcar y desmarcar subtokens 0D.
RF-5	El usuario podrá crear constraints 1D
RF-6	El usuario podrá crear constraints 2D
RF-7	El usuario podrá añadir un subtoken a un constraint 1D o 2D.
RF-8	El usuario podrá modificar el nombre de un constraint 1D o 2D.
RF-9	El sistema mostrará la posición de un subtoken 2D.
RF-10	El sistema mostrará todos los constraints relacionados con un subtoken 1D y 2D.
RF-11	El sistema mostrará el nombre y el id de sesión de los subtokens 1D y 2D.
RF-12	El sistema mostrará el valor de un subtoken 1D.
RF-13	El usuario podrá seleccionar posiciones en el mapa para crear un constraint 2D.
RF-14	El usuario podrá seleccionar rangos de valores para crear un constraint 1D.
RF-15	El usuario podrá modificar cualquier campo de un constraint 1D o 2D.
RF-16	El sistema almacenará todos los subtokens y constraints relacionados con un token.
RF-17	El sistema mostrará los constraints de un token 2D mediante una figura geométrica que representará el área de constraint.
RF-18	El sistema podrá importar constraints asociados a tokens y subtokens mediante la lectura de un fichero XML.
RFN-01	La aplicación funcionará en ordenador de escritorio.
RFN-02	La aplicación funcionará en un entorno interactivo basado en el paradigma de la computación ubicua.

RFN-03	La aplicación se basa en el paradigma WIMP
RFN-04	Deberá integrarse en el toolkit JUGUEMOS

Tabla 3: Tabla de requisitos funcionales y no funcionales

3.3 Casos de uso

Una vez recogidos los requisitos que debe satisfacer la aplicación, se ha procedido a la realización de un diagrama de casos de uso para poder observar qué acciones puede realizar nuestro usuario con relación a la interacción con el sistema. Una vez recogidos todos los casos de uso, se han especificado de forma individual, recogiendo su flujo de eventos natural y alternativo, si hubiese.

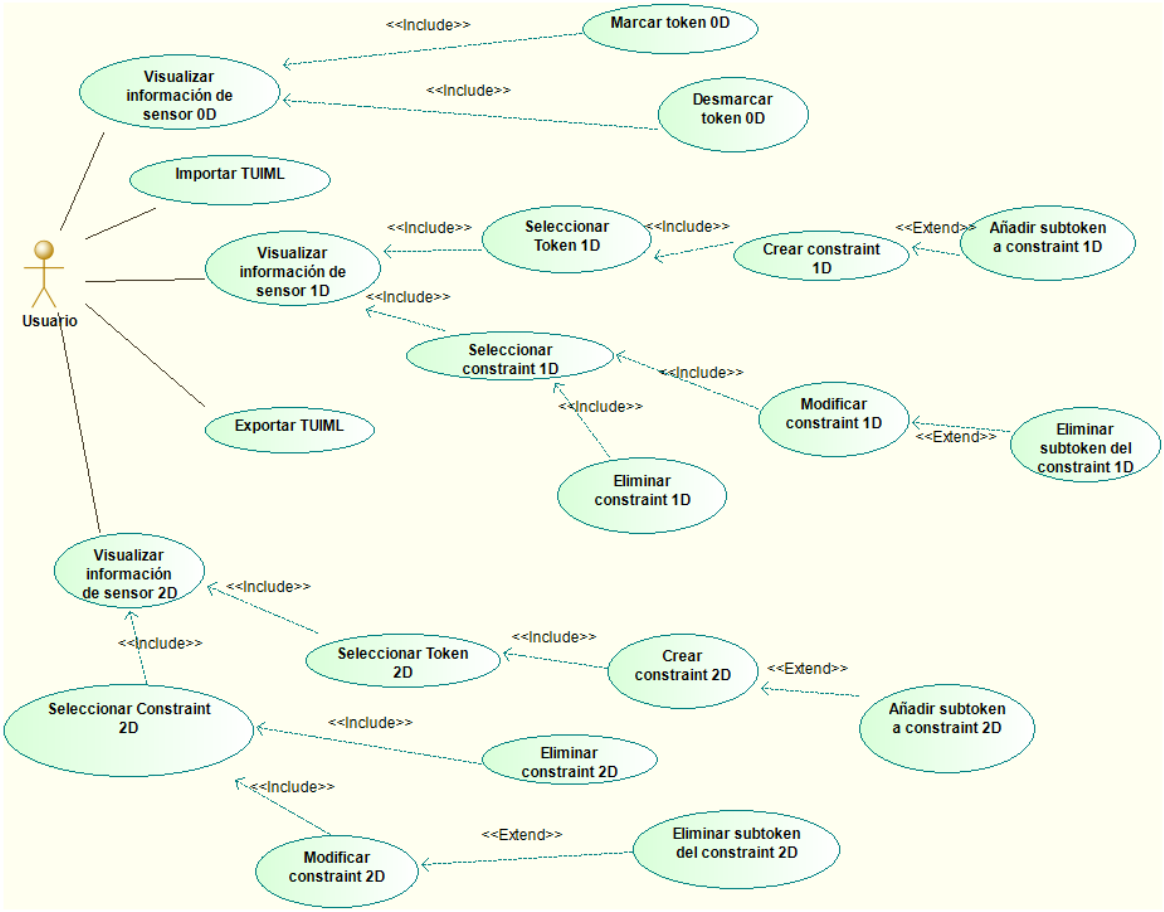


Imagen 9: Diagrama de casos de uso

Los casos de usos especificados se pueden observar en el Anexo B.

4 Diseño del sistema

Partiendo del análisis previo al diseño del sistema, se especificaron unos diagramas de secuencia, que expresan cómo será la interacción y comunicación entre los diversos componentes del sistema en los casos de usos y requisitos recogidos.

Los componentes del sistema que especifican la interacción y paso del flujo de la información son: el usuario, los sensores, la interfaz gráfica, y el semántico. Dichos diagramas de secuencia se pueden observar en el Anexo A.

Una vez se realizó el análisis de la interacción que debería de tener la aplicación con el usuario, se dispuso a realizar un prototipo de la interfaz del proyecto, que se explicará en la siguiente subsección.

4.1 Diseño de la interfaz

La interfaz se ha diseñado en dos iteraciones, una primera donde representar la interacción básica del usuario con el sistema y una segunda donde se han pulido cuestiones de usabilidad.

Durante las primeras reuniones de diseño con los investigadores del proyecto JUGUEMOS y tras la familiarización con los objetivos del trabajo, se sugirió un primer prototipado de la interfaz, cuyas pantallas son mostradas en las imágenes 10 a 16.

Imagen 10: Menú principal del primer prototipo

En la imagen 10 se observa el primer acercamiento al menú principal del trabajo. En esta pantalla se muestra una lista de sensores con información sobre el mismo, y se ofrece un botón para importar TUIML. Con estas dos funcionalidades se satisfacen los requisitos funcionales 2 (El sistema muestra una lista de sensores conectados) y 18 (El sistema podrá importar constraints asociados a tokens y subtokens mediante la lectura de un fichero XML).

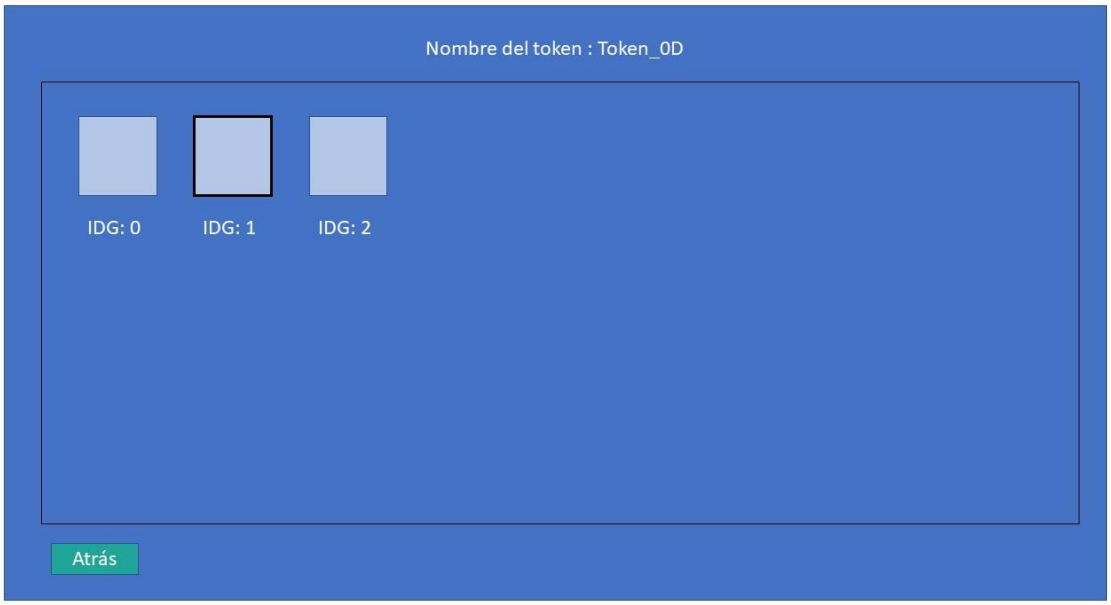


Imagen 11: Pantalla de constraints 0D

En la imagen 11 se observa la pantalla que aparece cuando un usuario selecciona un sensor 0D en el menú principal.

En esta pantalla se muestra una lista de los subtokens 0D que han ido llegando, los cuales tienen un borde negro si se han añadido al constraint o no. En esta pantalla se satisface el requisito 4(El usuario podrá marcar y desmarcar subtokens 0D).

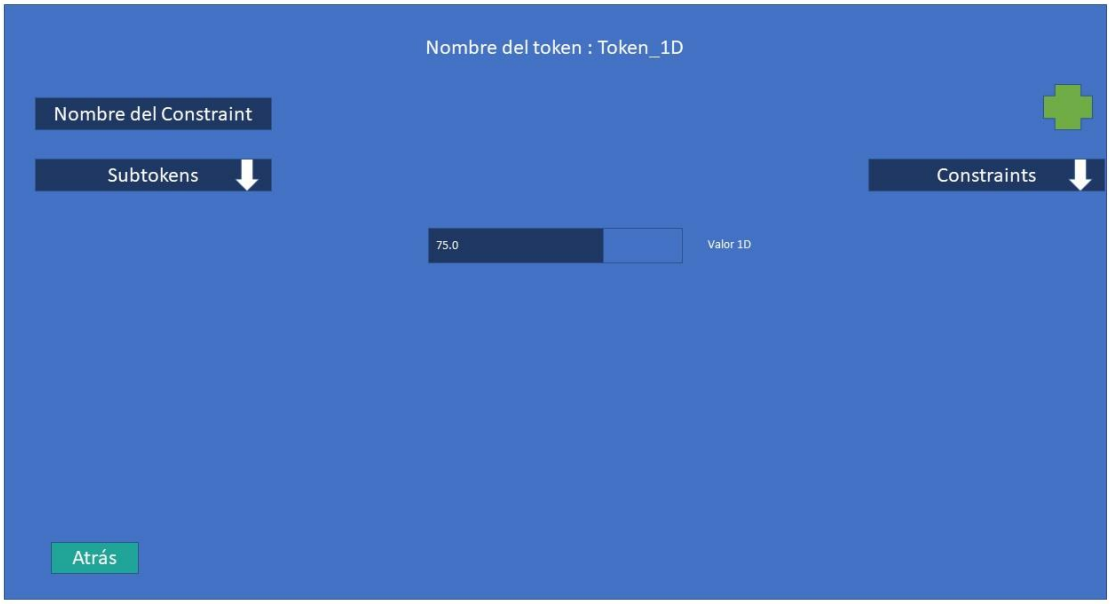


Imagen 12:Pantalla de constraints 1D

En la imagen 12 se muestra la pantalla que aparece cuando un usuario selecciona un sensor 1D en el menú principal.

En la parte central se observa una barra que toma valores desde el 0.0 hasta el 1.0, con dos decimales, y se rellena según el valor que reciba del sensor 1D.

En la parte izquierda se observa un campo de texto en el que se puede poner el nombre del constraint, y una lista que representa los subtokens que han ido llegando, satisfaciendo el requisito funcional 11 (El sistema mostrará el nombre y el id de sesión de los subtokens 1D y 2D).

En la parte derecha hay un botón verde en forma de cruz, que sirve para añadir un nuevo constraint, y debajo del mismo, una lista de todos los constraint creados. El botón satisface el requisito funcional 5(El usuario podrá crear constraints 1D) y la lista de constraints el requisito 10 (El sistema mostrará todos los constraints relacionados con un subtoken 1D y 2D).

Imagen 13: Pantalla tras añadir un constraint 1D

La imagen 13 es la pantalla que aparece cuando un usuario crea un constraint 1D.

Si comparamos con la imagen 12, en la parte central ha aparecido una nueva barra horizontal debajo de la barra donde se representan los valores recibidos, que sirve para señalar el rango de valores del constraint, cumpliendo el requisito funcional 14 (El usuario podrá seleccionar

rangos de valores para crear un constraint 1D).

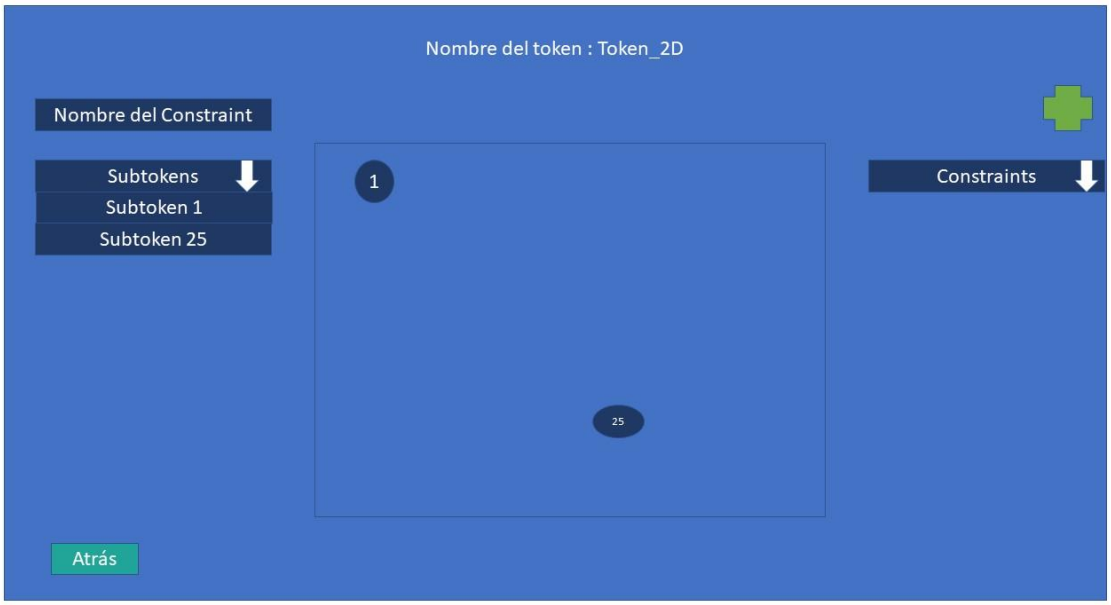


Imagen 14: Pantalla de constraints 2D

En la imagen 14 se observa la pantalla que aparece cuando un usuario selecciona un sensor 2D en el menú principal.

Los subtokens se representan con un círculo de color más oscuro que la silueta, y con el IDG en el centro de cada subtoken, cumpliendo también el requisito funcional 1.

Al igual que la pantalla de un sensor 1D, en la parte izquierda se distingue un campo de texto en el que se puede poner el nombre del constraint, y una lista que representa los subtokens que han ido llegando, y en la parte derecha aparece un botón verde en forma de cruz, que sirve para añadir un nuevo constraint, y debajo del mismo, una lista de todos los constraint creados. Estos tres elementos gráficos cumplen los requisitos 6 (El usuario podrá crear constraints 2D), 7, 10 y 15 (El usuario podrá seleccionar posiciones en el mapa para crear un constraint 2D).

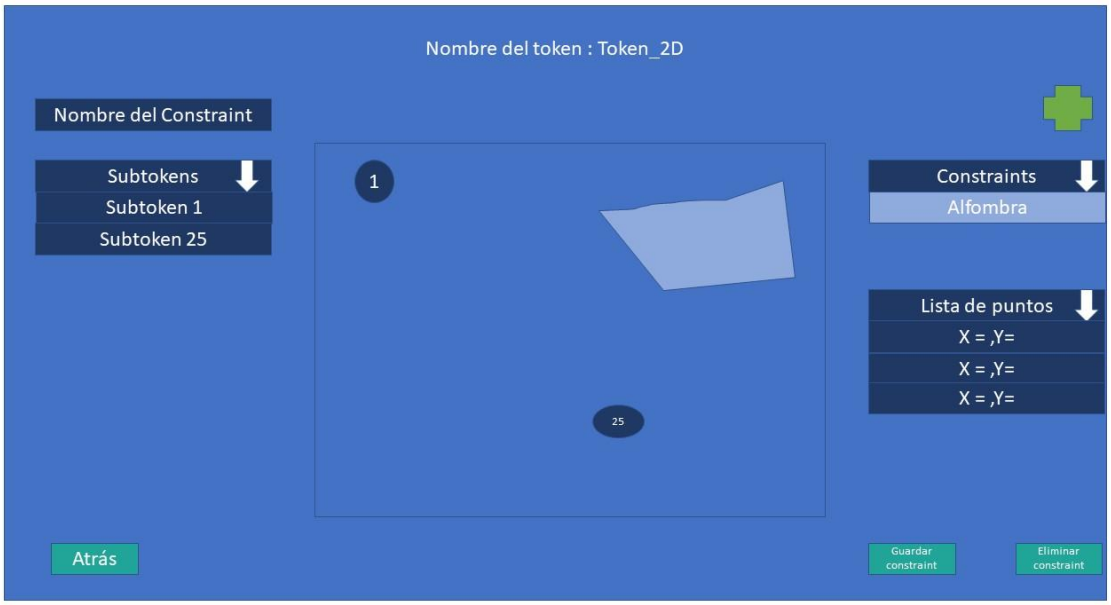


Imagen 15: Pantalla de constraints 2D tras añadir un constraint al token

En la imagen 15 se muestra la pantalla tras haberse creado un constraint que pertenece al token. El constraint es dibujado como un polígono con un número de vértices definidos por el usuario. Con esto se cumple el requisito 17 (El sistema mostrará los constraints de un token 2D mediante una figura geométrica que representará el área de constraint). La creación del constraint satisface el requisito funcional 13 (El usuario podrá seleccionar posiciones en el mapa para crear un constraint 2D).

La primera iteración de la interfaz se diseñó teniendo en cuenta los principios de diseño de las interfaces gráficas:

- Sabiendo que los usuarios de la aplicación son desarrolladores que no tienen por qué ser expertos en el dominio de desarrollo de juegos pervasivos o aplicaciones basadas en computación ubicua, se ha diseñado la interfaz pensando en: Se ha dividido la pantalla de forma que en el centro de las pantallas se pueda observar la información importante, como los subtokens o constraints, y se han dejado los laterales de la pantalla para información menos importante.
- Se ha mantenido un patrón constante en el uso de elementos gráficos como listas desplegables o botones, se reutilizan en todas las pantallas para que el usuario se acostumbre a su uso.
- La información se agrupa según su naturaleza, por ejemplo, la información de los constraints como la lista de constraints y la lista de los puntos de los constraints se muestran cerca uno de otro.
- Se asocia un color a cada elemento gráfico, por ejemplo, los botones de interacción con los elementos TUIML son de color verdoso, y el botón de importación de TUIML tiene un color más azulado.
- Se respeta el uso de los espacios en blanco para agrupar los elementos, dejando margen entre elementos gráficos.
- Se han seleccionado colores más oscuros para retroalimentar al usuario, por ejemplo, cuando un usuario seleccione un subtoken 2D, dicho subtoken se oscurecerá para poder distinguirlo del resto e indicar al usuario que se ha seleccionado.

- Se han escogido iconos que representen acciones del mundo real y le sea más fácil al usuario saber cuál es la acción que realiza el botón. Por ejemplo, con el botón de añadir constraint.

4.2 Arquitectura del sistema y vista de módulos

En esta sección se va a presentar la organización del sistema en módulos, ya que es un sistema que realiza tareas distintas, y es conveniente explicar cómo se han organizado las mismas en módulos y cuál es el alcance de cada uno.

Para el diseño de este proyecto software se ha elegido como punto de partida el patrón arquitectural *Modelo-Vista-Controlador* (MVC), pero con varias modificaciones al mismo:

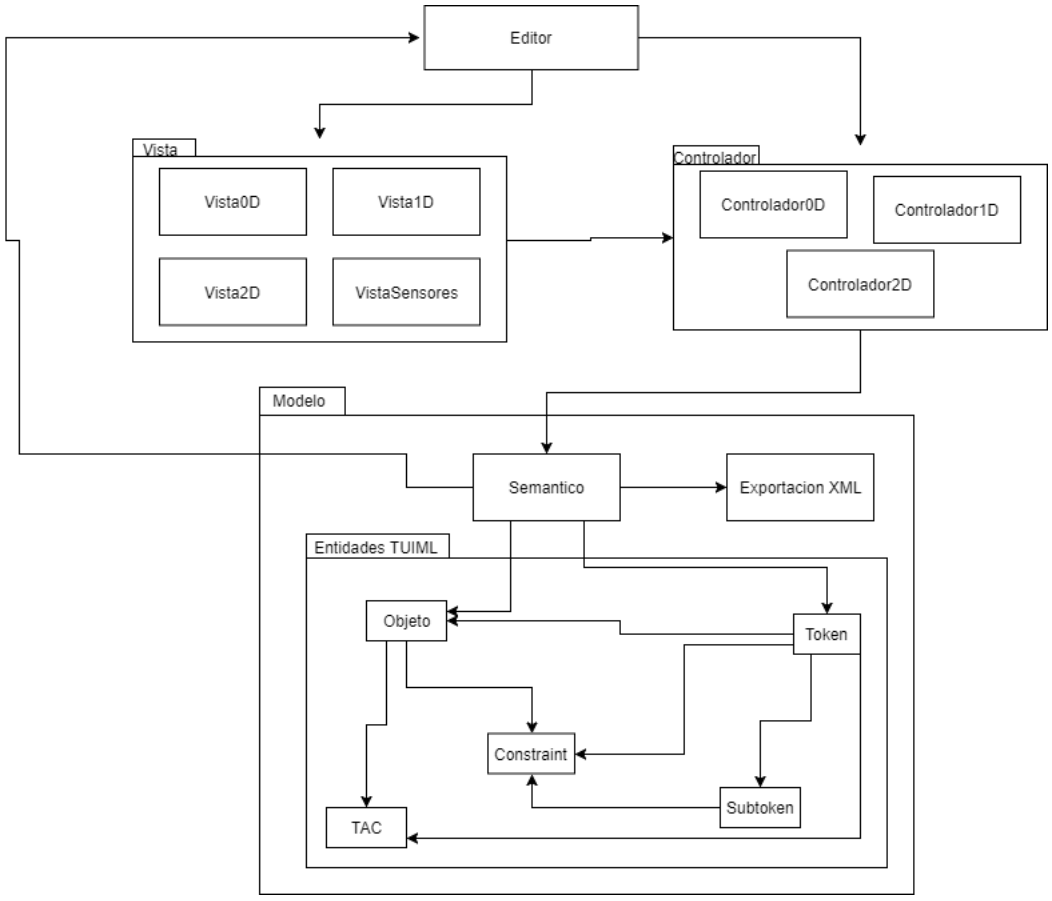


Imagen 16: Vista de módulos del sistema

Como se puede observar en la imagen 16, el sistema está compuesto, principalmente, de la interacción entre los módulos de tres paquetes y un módulo general que actúa de intermediario entre la interacción del usuario y la lógica del sistema.

La arquitectura del sistema está constituida por 4 integrantes fundamentales: la *Vista,* el *Controlador,* el *Modelo* y el *Editor:*

- La Vista: El paquete está formado por módulos que contienen los elementos gráficos que se mostrarán al usuario. Como hay distintos tipos de datos que puede enviar un sensor a la aplicación, se han creado distintos módulos para que cada uno albergue los elementos gráficos necesarios para representar los valores adecuadamente, dependiendo si los datos son de tipo 0D,1D o 2D.

Además, se ha incluido un módulo que contiene los elementos del menú principal, desde el cual el usuario podrá ver todos los sensores conectados, y acceder a otras opciones como puede ser la importación/exportación de ficheros TUIML.

- El Controlador: Los controladores son los módulos que se encargan de interpretar la interacción del usuario con los elementos gráficos de la vista, tales como clicks de ratón, e informar de los cambios al modelo. Los módulos se organizan según el tipo de sensor, es decir, para un sensor de tipo 1D, el controlador 1D estará al tanto de la interacción del usuario con los elementos propios de la vista 1D, además de incluir otras funciones necesarias para la correcta interacción.
  Los controladores son la parte del sistema que contiene un peso más grande de la lógica del sistema, puesto que además de controlar la interacción, debe controlar las restricciones asociadas al modelo y al dominio.
- El Modelo: El modelo es el integrante del sistema que contiene la parte de almacenamiento de datos, la parte donde se determinan los conceptos del dominio de la aplicación, y otros módulos como el de exportación a xml, o de recepción/envío de mensajes.
  El modelo incluye varios módulos que representan objetos del dominio, que a su vez representan ideas que se utilizan en el mundo real, y todas las relacionas entre ellas. Conceptos de los que se han hablado, como *Token*, *Constraint*…
  La parte fundamental y más importante del modelo es el módulo que se conoce como *Semántico*, un componente que actúa almacenando la información seleccionada por el usuario (*Tokens, Objetos, Subtokens…)*, enviándola a las Vistas*,* o cambiando su contenido según lo que informe el controlador. Además, el Semántico se encarga del tratado y envío de los mensajes en formato OSC, tanto en recibir los mensajes de los sensores e informar a las Vistas, o enviar mensajes TAC a los Hosts.
- El Editor: El editor funciona como un intermediario entre la Vista y el Controlador, redirigiendo el flujo de información del sistema desde la interacción del usuario hasta la respuesta del sistema al mismo.

Una vez se ha diseñado la estructura del sistema y la interfaz gráfica, se va a proceder a explicar la implementación de estos diseños, además de explicar los problemas que han surgido a lo largo del proceso de implementación.

5 Implementación del sistema

Para la implementación del sistema se ha partido del código ya funcionando del toolkit JUGUEMOS. El proyecto se construye sobre la base del toolkit y las clases ya implementadas, por lo que el editor gráfico desarrollado se debe integrar con el software del Broadcaster ya existente y varias decisiones de implementación se han tomado de acuerdo con el estado previo del toolkit.

El Broadcaster está codificado con Processing [PRO], un lenguaje de programación basado en Java, y orientado a proyectos multimedia e interactivos de diseño digital. El toolkit utiliza una librería ofrecida por Processing para el envío y recepción de mensajes OSC la librería oscP5 [OSC].

Para representar los elementos gráficos de la interfaz se ha utilizado la librería ControlP5 [CP5], una librería que proporciona variedad de elementos gráficos, como *sliders,* botones o campos de texto, y que además incluye controladores para los diversos elementos. Se han evaluado otras alternativas de librerías de controladores GUI, como G4P [G4P], pero se ha elegido CP5 porque contiene una documentación más extensa y variada, siendo por esto la más utilizada para la creación de GUI.

5.1 Diagrama de clases

En esta sección se va a presentar el diagrama de clases de la interfaz gráfica y las relaciones entre ellas (Imagen 28):

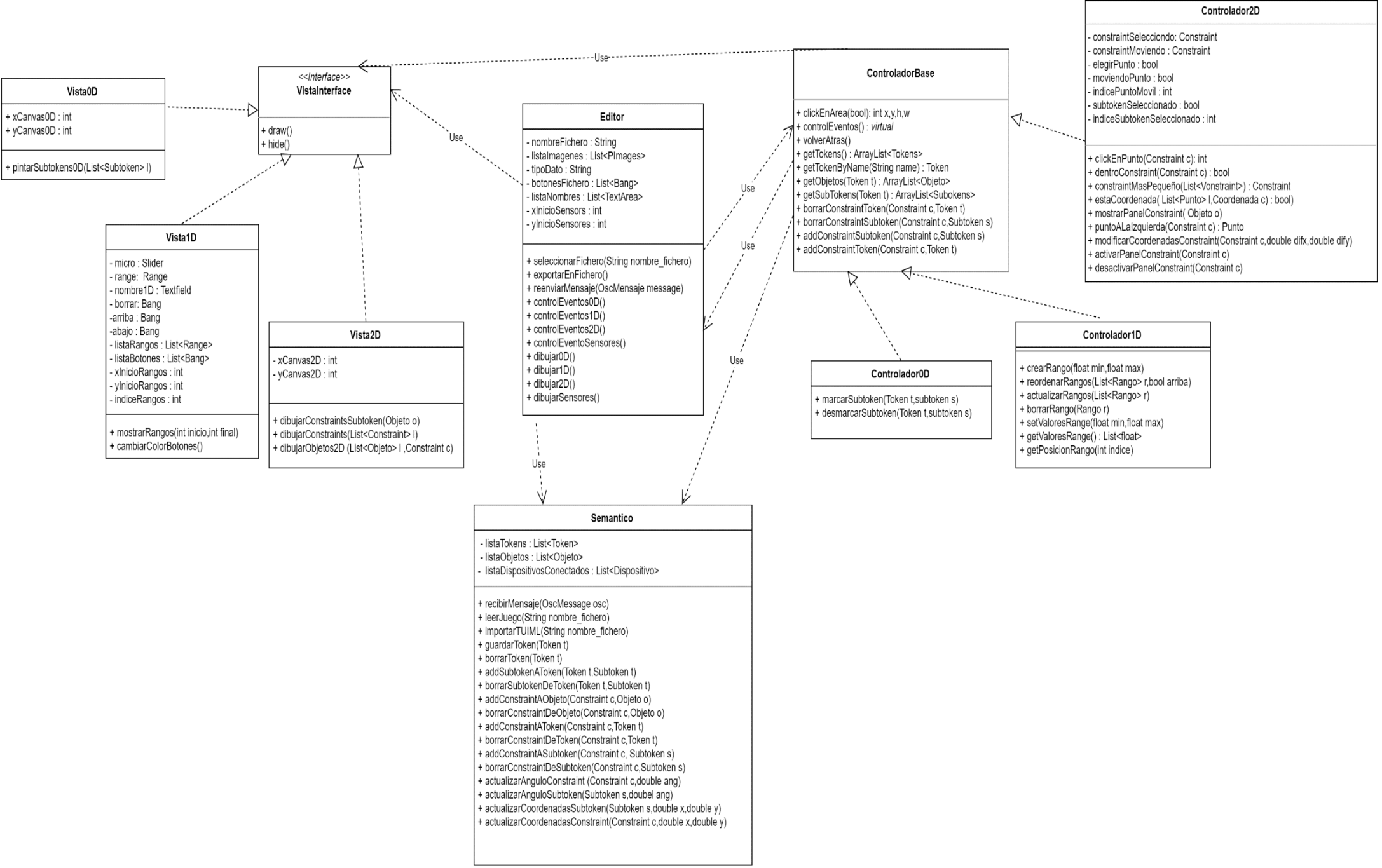


Imagen 17: Diagrama de clases

Se sigue manteniendo la idea de un diseño MVC, en el que el Editor muestra la información de la Vista correspondiente, y trata la interacción con el usuario mediante el Controlador asociado a la Vista, y si ha habido algún cambio, se modifica en el Modelo (en el proyecto la clase Semántico) y finaliza la interacción modificando los campos de la Vista.

Se va a proceder a explicar las clases de forma resumida con sus atributos y funciones más relevantes en el anexo E.

5.2 Desarrollo iterativo de la interfaz

El desarrollo de la interfaz fue cambiando e iterando durante todo el desarrollo del proyecto. En concreto, hubo dos iteraciones con sesiones de prueba con usuarios para detectar fallos o mejoras en la interfaz gráfica del proyecto.

5.2.1 Primera iteración

La primera sesión de prueba se realizó durante los primeros compases del desarrollo del proyecto, ya que se disponía de una implementación del diseño presentado en la sección 4.1.

Se llevó a cabo una prueba preliminar con usuarios para poder detectar fallos de implementación en el proyecto, y si había alguna cuestión de la interfaz que dificultase su uso.

Esta primera sesión tuvo lugar el 13 de abril de 2018 y fue la primera toma de contacto con usuarios reales. Los usuarios fueron los 6 estudiantes de la asignatura del máster en Ingeniería Informática de la Universidad de Zaragoza de código 62228, llamada Computación gráfica-entornos inmersivos-multimedia. Esta sesión de prueba, que duró 4 horas, se realizó en el espacio interactivo en Etopia, en el que se les explicaba el funcionamiento y estructura general del toolkit JUGUEMOS, y se les pedía que generasen un fichero TUIML utilizando la interfaz gráfica del proyecto.

La técnica de prueba utilizada es la conocida como *Thinking aloud* [BR00], un protocolo usado para reunir información en el campo del testeo de usabilidad de software, en el que se le encomienda una tarea a un usuario y éste tiene que intentar cumplirla, diciendo en voz alta lo que está pensando durante todo el proceso.

Esta sesión se diseñó en torno a una tarea general, que es crear el TUIML para un juego pervasivo concreto, el juego de la habitación o *kids room* [BID99], en el que un niño tiene que hablar con distintos muebles de su habitación para obtener una palabra mágica que le permita ahuyentar al monstruo que vive en su armario. Esta tarea general se ha simplificado y subdividido en dos subtareas para que pudiera ser realizada en el tiempo planeado para la sesión.

Dichas tareas fueron: *Crear dos constraints 2D, uno llamados "silla" que representase la posición de una silla dentro del espacio y "alfombra", que representase la posición de una alfombra dentro de un espacio*, y *Crear un constraint 1D llamado "grito" que represente el grito de una persona.*

Aunque durante la sesión todos los usuarios participantes lograron acabar todas las tareas, hubo algunas tareas que les supuso más complicadas que el resto, que era poder crear un constraint de tipo 2D. 5 de los 6 usuarios no entendían cómo interactuar con los elementos de la pantalla, y esto hacía que se sintieran desorientados y sin tener una idea clara de cómo realizar los pasos necesarios para cumplir la acción(seleccionar botón de crear constraint -> seleccionar puntos -> guardar -> renombrar) necesitados de ayuda para continuar. El otro usuario completó todas las tareas sin dificultad.

Tras la sesión con los usuarios se recogieron estos aspectos respecto a los problemas de usabilidad de la interfaz:

- Los usuarios no sabían cómo añadir un constraint a un token, se sentían perdidos porque los botones que había en esa versión no incluían texto que ayudase a dar información sobre las acciones del botón. Además, en la pantalla del sensor 2D la primera tendencia de los usuarios era presionar la pantalla para crear un constraint, no buscaban pulsar ningún botón.
- Los usuarios no sabían cuál era el fichero TUIML o si habían seleccionado alguno.
- Los usuarios no sabían cuándo se había acabado el proceso de creación de constraints, porque no había ningún cambio visual que lo indicase.
- Los usuarios no encontraban en el menú principal cuales eran las opciones para la gestión de ficheros TUIML.
- Los usuarios no sabían cuál era el token o subtoken activo dentro del sensor 2D, cuando alguno de ellos seleccionaba un subtoken, la interfaz no indicaba visualmente de forma clara que habían seleccionado otro subtoken o token.
- Los usuarios no tenían claro de forma visual donde se guardaban los constraints 1D. Cuando el usuario había acabado de crear un constraint 1D, no sabían si su acción había tenido algún efecto porque la interfaz no comunicaba nada.
- Los usuarios no sabían cómo exportar un fichero, ya que en esta versión no se podía crear un fichero desde dentro de la aplicación, y los usuarios intentaban buscar alguna opción para ello.

5.2.2 Segunda iteración

A raíz de los resultados obtenidos en la prueba de la primera iteración de la interfaz, se diseñó e implementó una segunda versión de la interfaz. La implementación de la interfaz se muestra en las imágenes 18 a 25.

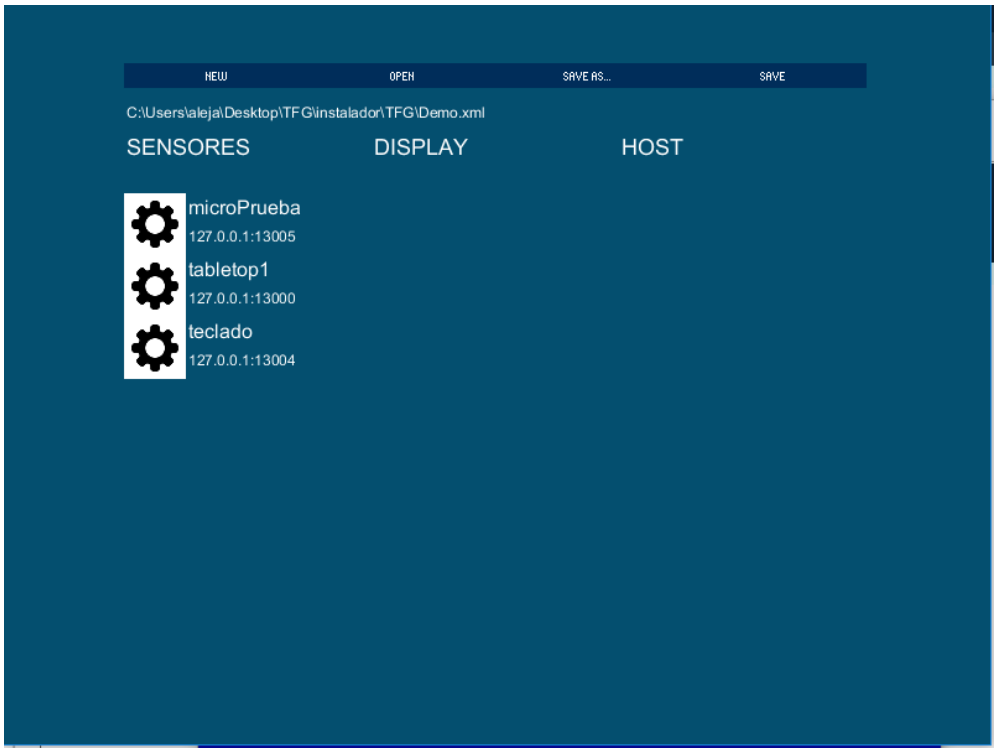


Imagen 18: Menú principal de la segunda versión de la interfaz

En la imagen 18 se observa cómo es la versión final del menú principal del proyecto.

Se han cambiado la disposición horizontal de los cuadrados, por 3 listas verticales, en la que se indican los sensores, los dispositivos de display, y los hosts, cada uno en su lista.

Los elementos de la lista se representan con su nombre, IP del dispositivo, y puerto de conexión, y si el dispositivo es un sensor, se acompaña con una imagen de un engranaje que sirve para ver los mensajes que se están recibiendo.

Además, se ha incluido una barra de menús con 4 opciones facilitando así su uso y debido a problemas de usabilidad en la anterior versión del prototipo. Este tipo de control es muy utilizado en el diseño de interfaces para organizar en el mismo espacio diversas opciones de gestión de objetos, en este caso el fichero TUIML activo. Se puede abrir un nuevo fichero(open), guardar uno previamente abierto(save), y guardar uno en un fichero a elegir por el usuario (save as). Además, se puede borrar cualquier modificación al fichero y empezar de cero (new).

Debajo de las opciones aparece una cadena de texto con la ubicación del fichero abierto, si es que el usuario ha seleccionado alguno. En caso contrario, aparece la cadena "*undefined*". Con esto proporcionamos *feedback* al usuario para que así cuando realice alguna acción reciba información de parte de la interfaz.

Imagen 19: Pantalla que se muestra al seleccionar un sensor 1D.

En comparación con la primera versión, se ha dividido la pantalla en dos zonas generales, la parte inferior contiene una lista de todos los constraints 1D, cada uno representado con una barra horizontal que identifica los dos valores del constraint con un color azul más claro. A la derecha de la barra está su nombre, y si se pulsa, a la derecha aparece la posibilidad de modificar el nombre, y a la izquierda un botón de borrado. Si hay más de 5 constraints en la

lista, se puede hacer *scroll* pulsando los botones a la derecha de los constraints, para subir y bajar de 5 en 5 el número de constraints.

En la parte superior se incluye una barra de nombre *valor1D* que informa al usuario de los valores recibidos del sensor 1D. Si el usuario pulsa dicha barra, aparece una justo debajo para que el usuario elija el rango de valores de los que se compone el constraint.

Antes de dichas barras, se encuentra el nombre del sensor 1D, y a su derecha hay una lista horizontal de botones que representan los subtokens conectados, a los cuales se pueden asignar los constraints. Si se pulsa un constraint que tiene asociado dicho subtoken, el botón aparece con un fondo negro, y morado en caso contrario.

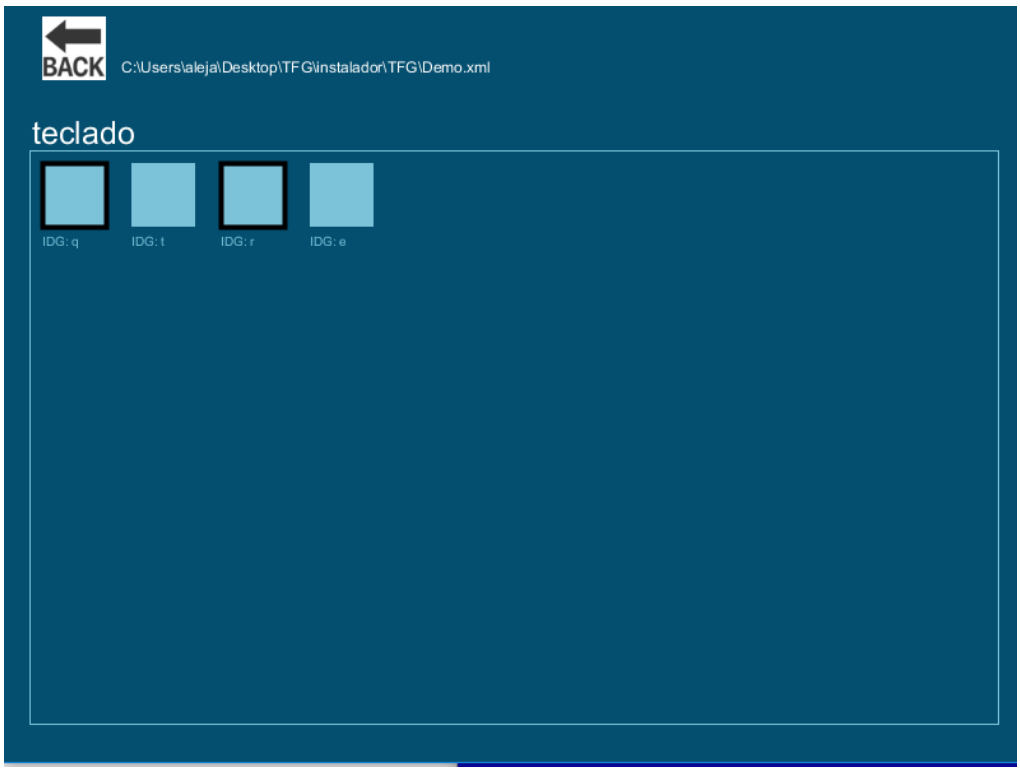


Imagen 20: Pantalla que representa la información del sensor 0D.

No ha habido cambios con la versión inicial.

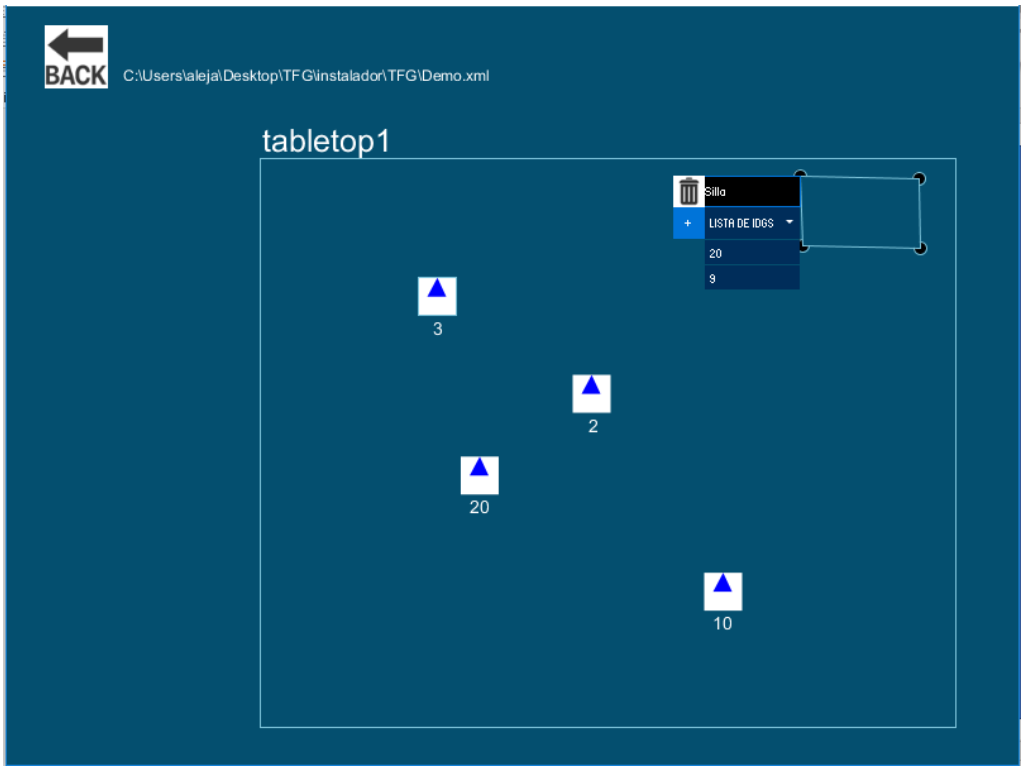


Imagen 21: Pantalla que muestra la información recibida por el sensor 2D.

Aunque la idea de representación de los tokens y constraints 2D no ha variado desde la primera versión, sí que se ha modificado la disposición de los elementos: No hay listas ni de constraints ni de puntos ni de subtokens, todo se ha eliminado para que el *focus* del usuario sea en la silueta cuadrada del centro, como se aprecia en la imagen 21. Esta decisión se ha tomado debido a las acciones de los usuarios recogidas en la prueba, la acción que hicieron todos ellos no fue buscar un botón para crear un constraint, directamente iban a tocar la silueta del centro.

Los subtokens se han representado con un cuadrado blanco y con un triángulo azul que representa la orientación del mismo, además de una cadena con su IDG.

Los constraints ahora tienen un panel de elementos gráficos, compuesto por un campo de texto para poder cambiar el nombre, un botón de borrado, un botón para asignar subtokens al constraints, y una lista de los IDGS de dichos constraints.

Los vértices están representados por puntos negros cuando el constraint no se ha seleccionado (se ha hecho click dentro de su área), y blancos cuando el constraint se ha seleccionado.

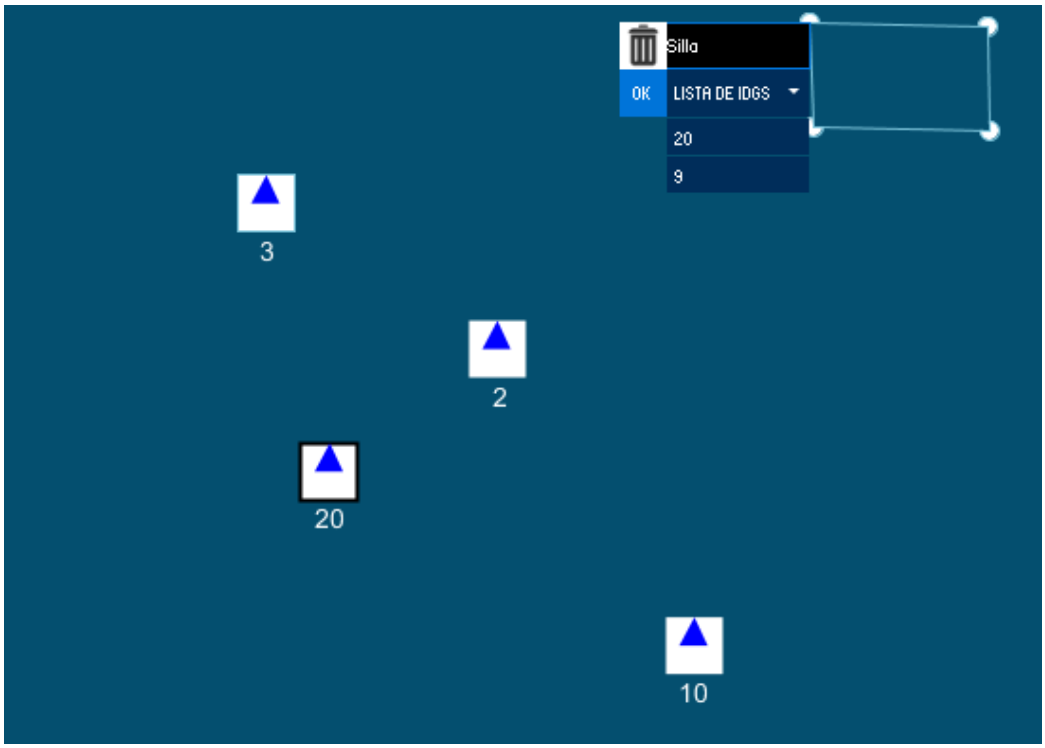


Imagen 22: Zoom de la pantalla de un sensor 2D cuando se pulsa el botón de añadir subtokens a un constraint

En la imagen 22 se aprecian los cambios gráficos en la interfaz que ocurren al pulsar el botón de añadir un constraint.

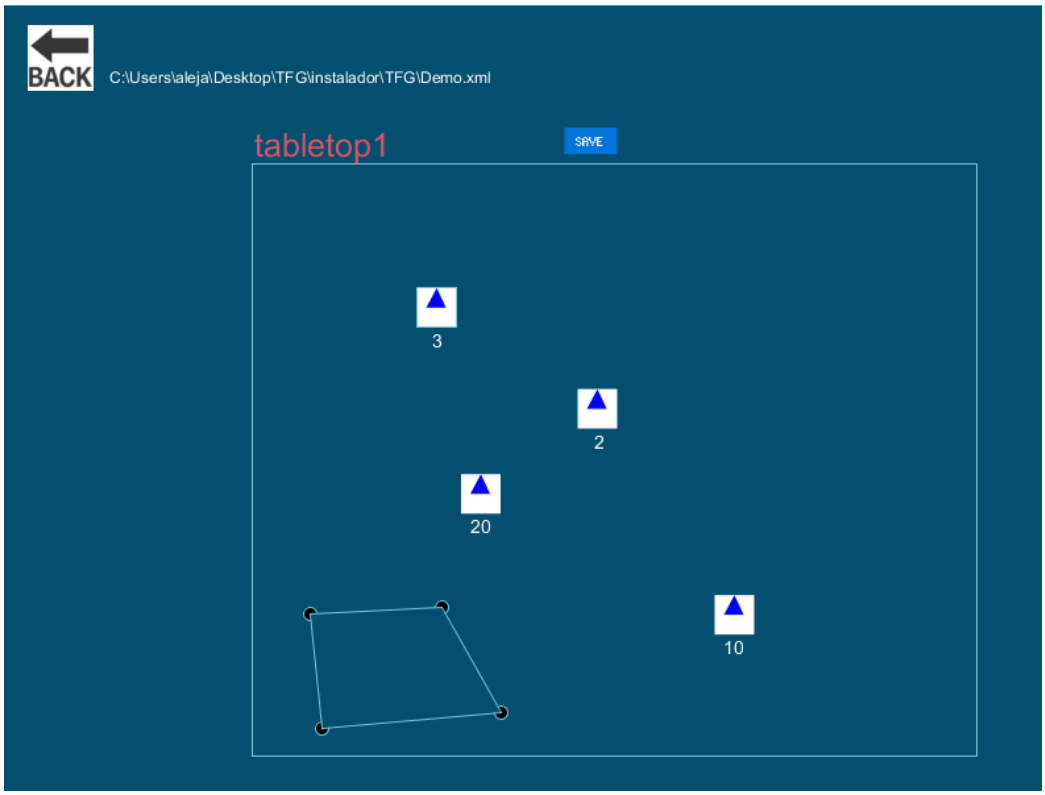


Imagen 23: Pantalla que aparece cuando el usuario añade un constraint asociado al token

En la imagen 23 se observa cómo reacciona la interfaz al añadir un constraint asociado al token.

Para ello, el usuario toca en cualquier punto de la silueta cuadrada, y automáticamente el programa empieza a contabilizar y guardar los vértices del constraint. Si el usuario quiere añadir un vértice, pulsa el botón izquierdo del ratón dentro del espacio de la silueta, y si quiere borrar un vértice ya existente, pulsa con el botón derecho sobre el vértice. Como se aprecia, la interfaz ofrece *feedback* al usuario al cambiar el color del nombre del token, y al hacer aparecer el botón de guardado.

Para terminar, el usuario pulsa el botón de *save*, y el constraint se guarda.

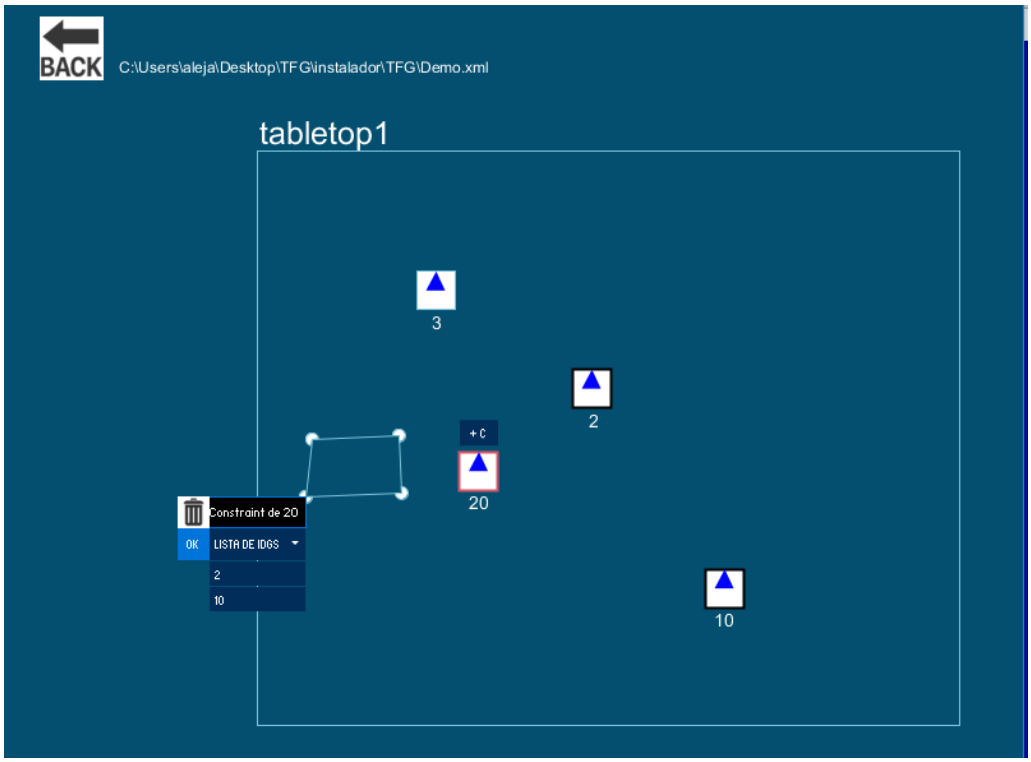


Imagen 24: Estado de la interfaz al seleccionar un subtoken

Como se observa en la imagen 24, un constraint, al ser seleccionado mediante un click izquierdo de ratón, es rodeado por una silueta rosada, y aparece un botón encima del subtoken, que al ser pulsado permite al usuario crear un constraint asociado al subtoken.

El proceso de adición, y elementos del panel de un constraint asociadas al subtoken mantienen las mismas propiedades que uno asociado a un token.

Además, como se observa en la imagen 25, el constraint rota conjuntamente con el subtoken, actualizando así, en tiempo real y de forma visual, la información enviada por el gestor del sensor asociado al token visualizado.

Imagen 25: Rotación de un subtoken

Para comprobar el funcionamiento de esta iteración del desarrollo de la interfaz en su contexto de uso real, se concretó una reunión de prueba para el 25 de mayo de 2018 con dos investigadores del AffectiveLab, conocedores del funcionamiento del espacio interactivo y del toolkit JUGUEMOS, y de las funcionalidades y objetivos del proyecto. Se llevó a cabo una sesión de demostración del producto en las que se buscaba obtener una valoración sobre la interfaz con usuarios avanzados. La reunión se trató sobre la funcionalidad del sistema para que ellos detectasen alguna inconsistencia y detalles de usabilidad en la interfaz. La entrevista se estructuró en dos partes principales:

- Se repasan los requisitos funcionales en diversas agrupaciones (véase la sección 3.2), dependiendo del tipo de dato 0D,1D o 2D.
- Se ejecutan los requisitos obtenidos y los investigadores interrumpen la demostración si hay alguna cuestión de interacción que consideran errónea.

Se sacaron estos resultados:

- La lista de IDGs de los paneles asociados a un constraint no estaban desplegadas por defecto, había que desplegarlas manualmente, por lo que los expertos no podían observar las adiciones o borrados de subtokens a un constraint.
- Cuando un usuario seleccionaba un subtoken, no sabía cuál era el token activo, similar a lo ocurrido en la primera reunión de evaluación.
- No había una coherencia con el uso de colores y sombreados en las opciones e interacciones con el sistema.
- Algunos colores de cadenas de caracteres dificultaban su visualización.

Para arreglar las cuestiones de usabilidad recogidas en esta última prueba, se implementaron estos cambios:

- Se implementó la idea de diseño para mostrar al usuario toda la información posible, de manera no intrusiva, y que fuera el usuario quien decidiera cuanta información quiere visualizar. Esto está relacionado con la lista de IDGs de los paneles asociados a un constraint, se dejan desplegadas por defecto, y cada vez que el usuario borra o añade un subtoken a un constraint, se muestran todos los cambios, y no como antes, que no se apreciaban bien.
- Se ha mantenido un estándar de colores, para que el usuario identificase cada color con un estado del sistema, por ejemplo, se ha utilizado el naranja para mostrar la llegada de mensajes en el menú principal, el blanco para mostrar cadenas de caracteres o números, y el rosa para señalizar que un token o subtoken ha sido seleccionado.
- Se ha cambiado la paleta de colores para que no fuesen difíciles de distinguir. Por ejemplo, se ha optado por utilizar un color de letra blanco porque era más sencillo de leer que una cadena con un color negro.

Con estos cambios realizados a la interfaz tras la segunda sesión de pruebas, se decidió que esta versión de la interfaz sería la definitiva.

5.3 Cuestiones y problemas de implementación

En esta sección se van a comentar algunas cuestiones y problemas de implementación que han ido surgiendo durante el desarrollo del proyecto. Como ya se ha indicado, la implementación e integración de la GUI parte del código del Broadcaster del toolkit JUGUEMOS, por lo que al integrar una GUI también se ha tenido que gestionar la información en las capas interiores, lo cual ha supuesto varios problemas:

- El Broadcaster recibe mensajes OSC de diversos dispositivos, y también puede exportar información TUIML de un fichero XML. Al añadir la GUI, se ha tenido que implementar teniendo en cuenta que la información va a provenir de tres fuentes de información:
    - De los mensajes *add,move* y *rotate* que recibe de los sensores.
    - Del lenguaje TUIML que exporta del fichero XML.
    - De las manipulaciones que realice el usuario en la interfaz: creación/borrado de constraints, asignación de subtokens a constraints…

  El tener que gestionar estos tres canales de información ha supuesto:

    - Codificar varias funciones que permitan convertir la información de un canal a otro. Por ejemplo, al leer un fichero TUIML y almacenar las estructuras recién exportadas, hay que convertir las coordenadas de un constraint del fichero (cuyo valor va de 0.0 a 1.0) a un valor presentable en la interfaz. También ocurre lo mismo cuando hay que exportar un fichero, ya que hay que convertir de las coordenadas con las que se presentan los constraints en la GUI, a sus valores adecuados en el fichero TUIML.
    - Mantener un orden en el que la información es procesada, ha habido que evitar que se tratase información de dos o más canales a la vez, ya que esto conllevaba problemas de consistencia de la información. Por ejemplo, no se puede permitir que el usuario modifique un constraint asociado a un subtoken, cuando se reciben mensajes de movimiento de dicho subtoken, o no se puede exportar un fichero TUIML mientras se están recibiendo mensajes de los sensores, ya que la información no sería la más actualizada. Para ello se ha

codificado diversos estados en los que el Broadcaster únicamente procesa mensajes de una sola fuente de información.

- El código del Broadcaster no tenía en cuenta que se fuera a implementar una GUI, por lo que varias acciones de recepción de mensajes y de importación han tenido que ser modificadas:
    - Al importar un fichero TUIML, dichas estructuras no se mantenían almacenadas, por lo que, al exportarlas, si no se habían creado otra vez, se perdían y no se exportaban. Por ejemplo, si se importaba una estructura que definía diversos constraints sobre un subtoken, si dicho subtoken no reaparecía en el Broadcaster, esta información se perdía. Para ello se ha creado un sistema de *back-up* de estructuras para almacenar toda esta información si necesidad de que el usuario tenga que recrearla manualmente.
    - Los gestores de los sensores del espacio interactivo enviaban los mensajes de forma desordenada, puesto que había algunos que enviaban un *move* antes de un *add*, o un *add* seguido de un *rotate*. Esto, en las diversas evaluaciones supuso un problema, ya que, en el momento de mostrar la información por pantalla, había operaciones que se desestabilizaban y daban lugar a errores. Para ello tenido en cuenta estas posibles combinaciones de mensajes y tratarlas de manera individualizada.
- Al tener un modelo de datos que puede ser accedido a la vez por distintos procesos, se ha tenido que tratar la concurrencia al acceder a las listas del semántico. Para ello se han utilizado las funciones *noLoop()*, *loop()*, que paralizan el proceso principal permitiendo a los procesos poder acceder a las listas sin problemas.
- Aunque Processing es un IDE originado para los proyectos multimedia, las librerías de implementación de GUI no ofrecían las funcionalidades necesarias para este proyecto. En concreto, se han encontrado muchos fallos con el tratamiento de eventos de la librería CP5. En Processing, el tratamiento de eventos funciona mediante una cola con propiedades ejecutándose en un hilo distinto al proceso principal, y se tratan los eventos tras la finalización del proceso principal. El problema es que los eventos propios de Processing se solapaban con los eventos de los elementos gráficos de CP5, por lo que un click de ratón tratado por el programa principal tenía mucha más prioridad que un click de ratón sobre un cuadro de texto, por ejemplo.
  En este caso, cuando el usuario hace click sobre un cuadro de texto, el evento es tratado como si fuera un click de ratón en el proceso principal, anulando el tratamiento propio del cuadro de texto. Por ello se ha tenido que implementar manualmente muchas funciones de tratamiento de eventos.
- Otra de las limitaciones que se han encontrado en Processing es que es necesario que haya un módulo con la función *draw()* (la cual se llama cada número de frames que selecciona el usuario) y no es posible redirigir la ejecución a otro módulo. Además, el tratamiento de eventos aumenta mucho en dificultad si hay distintos módulos cada uno con un bucle de tratamiento de eventos, puesto que los tratamientos de eventos que se encuentran en el módulo principal son los que más prioridad tienen. Los eventos no sólo se reducen a eventos de usuario, sino también a los mensajes OSC recibidos.
  Debido a estos problemas, se decide crear un módulo que actúe de intermediario entre el usuario, y el resto de los integrantes del sistema. La traza de ejecución de estos módulos se representa en el siguiente diagrama de secuencia:

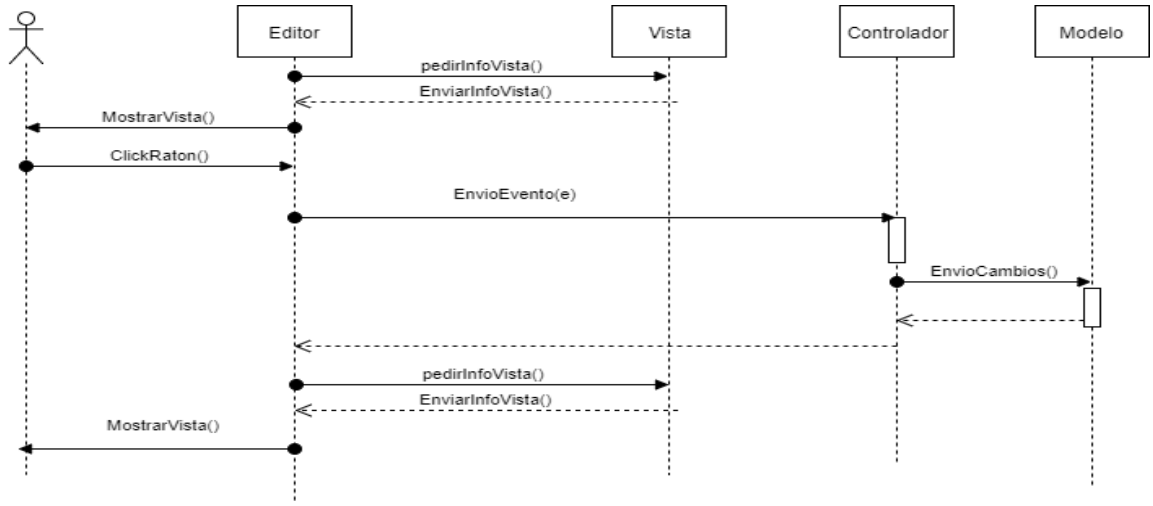


Imagen 26: Diagrama de secuencia que ilustra la interacción entre los distintos integrantes del sistema.

Este diagrama de secuencia ejemplifica la interacción entre el usuario y el sistema:

1. El Editor le envía un mensaje a la Vista para recibir la información de los elementos gráficos, y presentarlos por pantalla. La Vista a la cual el Editor envía los datos depende del tipo de sensor que esté conectado en ese momento.
2. El Editor recibe los elementos y los presenta por pantalla.
3. El Editor, recibe los eventos del usuario, como puede ser hacer click en el ratón, y se los redirige al Controlador pertinente.
4. El controlador trata esos eventos, para ver si dicho evento ha desencadenado algún cambio importante en el Modelo.
5. El Modelo modifica sus datos según la información enviada, y devuelve el flujo del programa al Controlador, y éste al Editor.
6. El Editor avisa a la Vista, y refresca la pantalla con los cambios efectuados.

Como se puede observar, este sistema tiene un módulo que actúa de intermediario, es decir, cualquier tipo de información que el sistema recibe, llega primero al Editor, el cual se encargará de repartir la información y de su tratamiento.

Esta implementación surge debido a la estructura anterior del toolkit JUGUEMOS, así como a la necesidad de adaptarse a un IDE como Processing, que no posee todas las herramientas para crear un proyecto de una manera más convencional, pero aun así se ha seguido la filosofía inicial del patrón MVC, aunque adaptándolo a nuestras necesidades.

6 Pruebas del sistema

En esta sección se muestran las pruebas de evaluación a las que se sometió el proyecto, y los resultados obtenidos en las mismas.

La evaluación tuvo lugar el día 4 de Julio en el espacio interactivo de Etopia, y su finalidad ha sido comprobar que se cumplen los requisitos funcionales, diseñando un conjunto de tareas que cubran todos los requisitos planteados en el proyecto.

Las tareas que se comprobaron en la evaluación fueron:

- La aplicación genera un fichero TUIML con sintaxis correcta y sin errores.
- El editor gráfico representa por pantalla correctamente todos los elementos del entorno sin errores.
- El Broadcaster envía mensajes TAC correctamente al proceso host tras importar el fichero TUIML.

Para las pruebas se crearon 4 ficheros TUIML utilizando el editor. Cada fichero representaba un tipo de sensor distinto: 0D,1D y 2D, todos deberían satisfacer las 3 condiciones arriba descritas.

6.1 Fichero 0D: ToyBox

Para comprobar que un sensor 0D funciona correctamente, se generó un fichero TUIML a partir de un dispositivo integrado en el espacio interactivo, el ToyBox [TOY], que permite integrar juguetes convencionales en el espacio interactivo. Para esta prueba se ha conectado toybox a una espada de juguete que tiene un botón, y el niño lo manipula pulsando dicho botón. Por tanto, se trata de un sensor 0D.

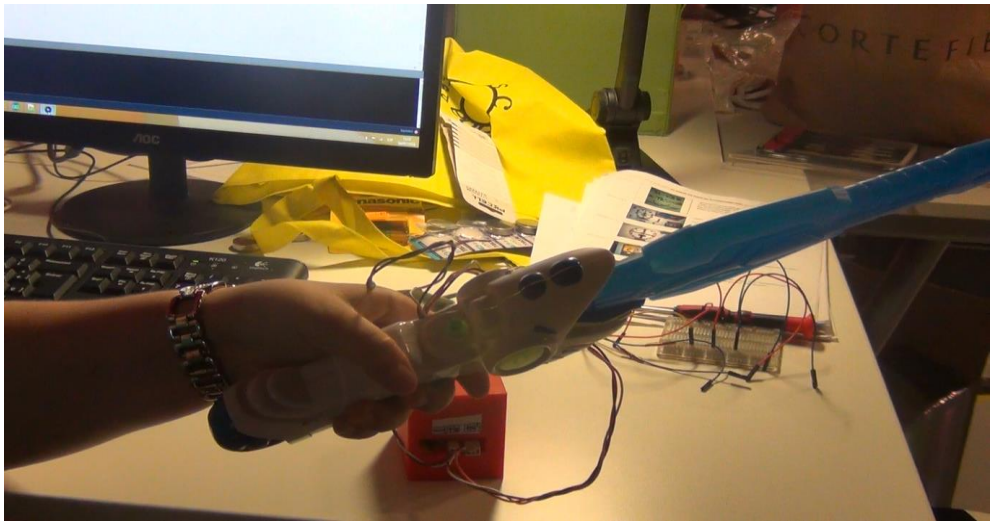


Imagen 27: Dispositivo ToyBox

En la imagen 28 se puede observar una imagen del dispositivo ToyBox conectado a la espada de juguete con el botón manipulable.

El proceso asociado al gestor envía al Broadcaster un mensaje cada vez que el botón es pulsado o despulsado. Para incorporar dicha manipulación al fichero TUIML hay que marcar en el editor el subtoken botón para añadirlo al constraint 0D (Imagen 29). Tras haber creado el fichero TUIML, se probó que en el proceso host se recibían los mensajes de botón pulsado y despulsando. (Imagen 3)

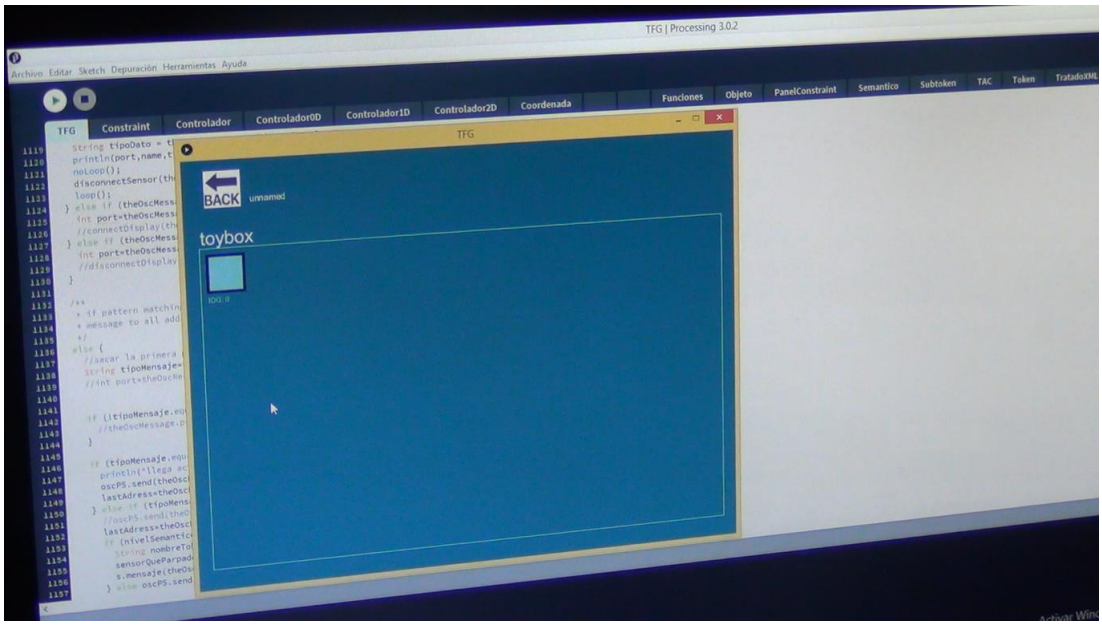


Imagen 28: Creación del constraint 0D en la sesión de evaluación.

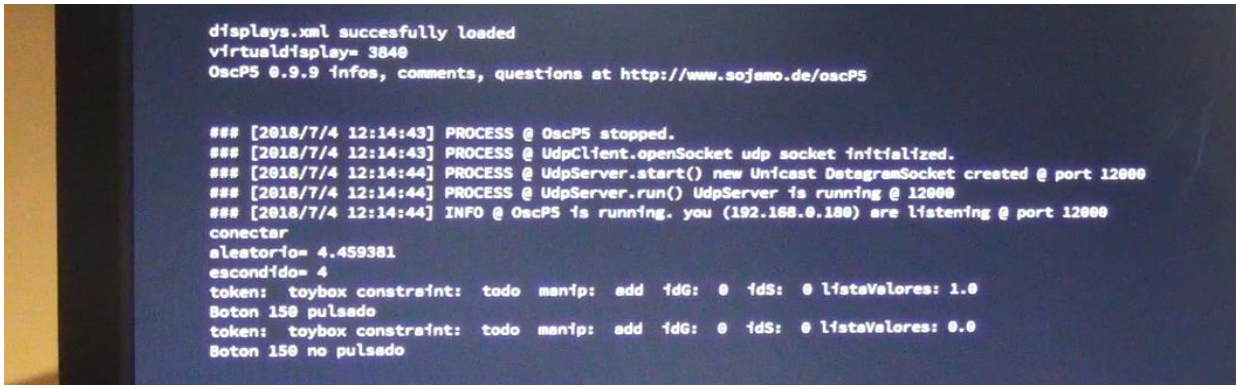


Imagen 29: Mensajes enviados por el Broadcaster al pulsar y despulsar el botón.

6.2 Fichero 1D: Micrófono

Para generar la manipulación con un sensor 1D se utilizó un micrófono del espacio interactivo, y se creó el constraint llamado *Grito (*Imagen 31*)* que registrase las frecuencias de voz por encima de un umbral determinado, y enviase mensajes al Host cada vez se superase ese umbral.

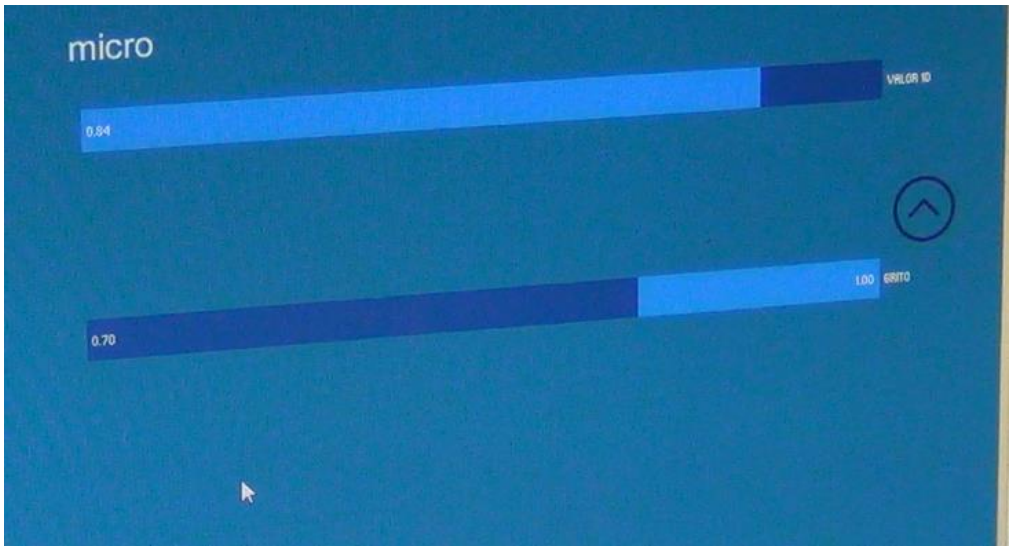


Imagen 30: Imagen ampliada de la creación del constraint 1D en el editor

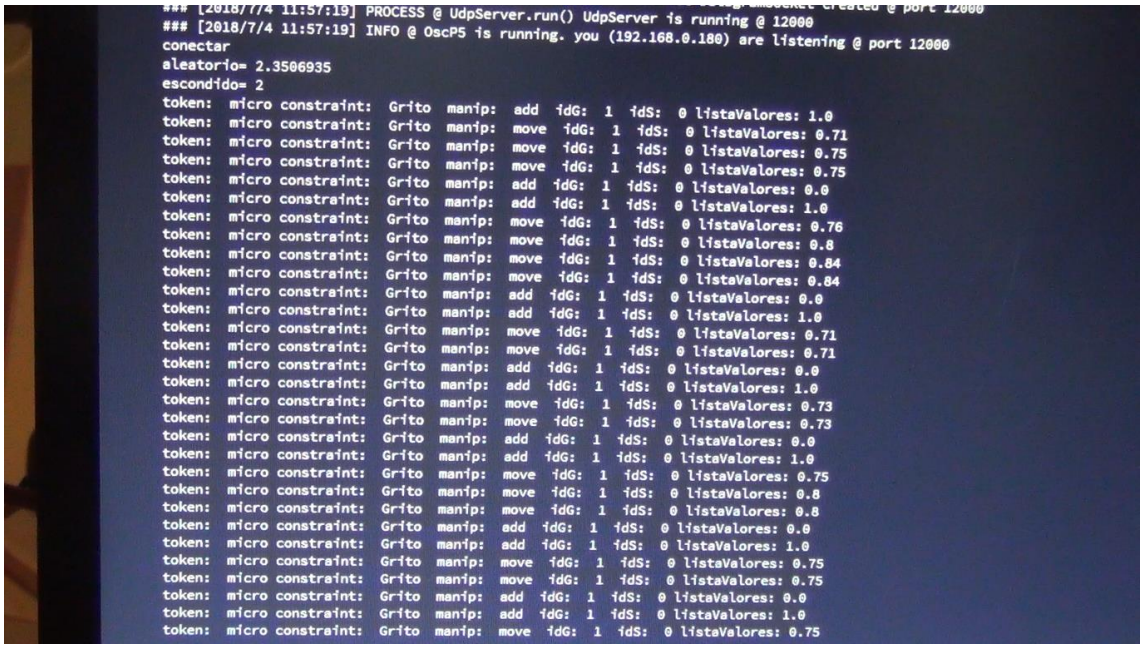


Imagen 31: Mensajes enviados por el Broadcaster cuando recibe valores mayores que el umbral superior del constraint Grito

En la imagen 32 se observa cómo llegan los mensajes: el token *micro* ha sufrido una interacción en el constraint *Grito,* la manipulación puede ser *add* si es el primer mensaje que recibe de una sucesión de mensajes (o el último) o *move* si el valor de la frecuencia del mensaje recibido es mayor que el umbral del constraint. Como se aprecia, todos los mensajes *move* que se reciben son de un valor mayor que 70, que es el límite superior del constraint *Grito.*

6.3 Ficheros 2D: RTLS y tabletop

Para la evaluación con sensores 2D se realizaron dos pruebas distintas, primero se crearon unas reglas TUIML para un juego de tabletop, y después para otro juego utilizando el sistema RTLS.

En el tabletop se generó un fichero TUIML para un juego de disparos, en el que un token, una nave espacial, tiene asociado un constraint que registra cada vez que la nave dispara (se pulsa el botón de disparo que hay en la nave). Además, también se han creado dos constraints asociados al tabletop, que dividen la superficie en dos zonas, derecha e izquierda, y que registran cada vez que la nave entra en alguna de las dos zonas.

En la imagen 34 se puede ver la nave puesta en el tabletop, y los dos constraints separando la superficie en dos partes.

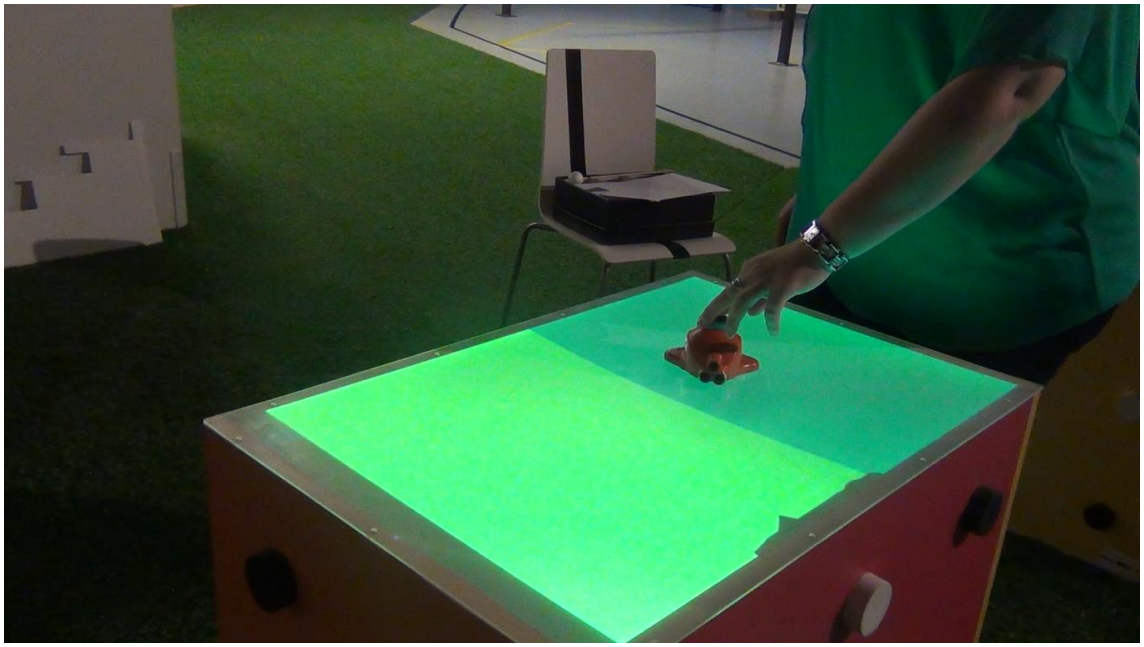


Imagen 32: Nave de disparo sobre el tabletop

En la imagen 33 se observa la creación de los dos constraints asociados al token en el editor, que dividen la mesa en mitad derecha y mitad izquierda.

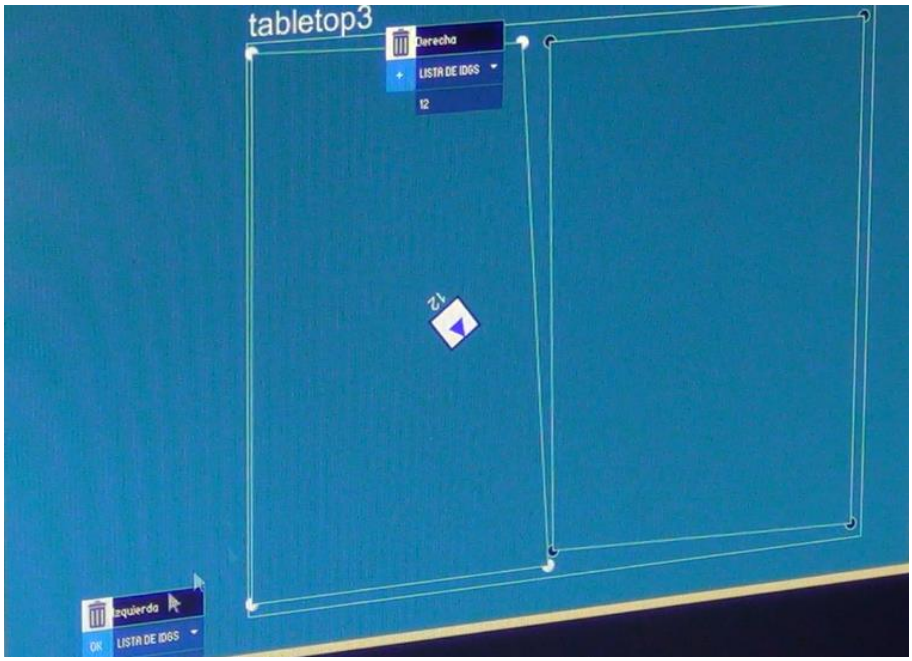


Imagen 33: Imagen ampliada de la pantalla del editor al crear los constraints "Derecha" e "Izquierda"

En la imagen 34 se observa, en el editor, la creación del constraint asociado al subtoken 12(juguete de nave espacial) para registrar el disparo.

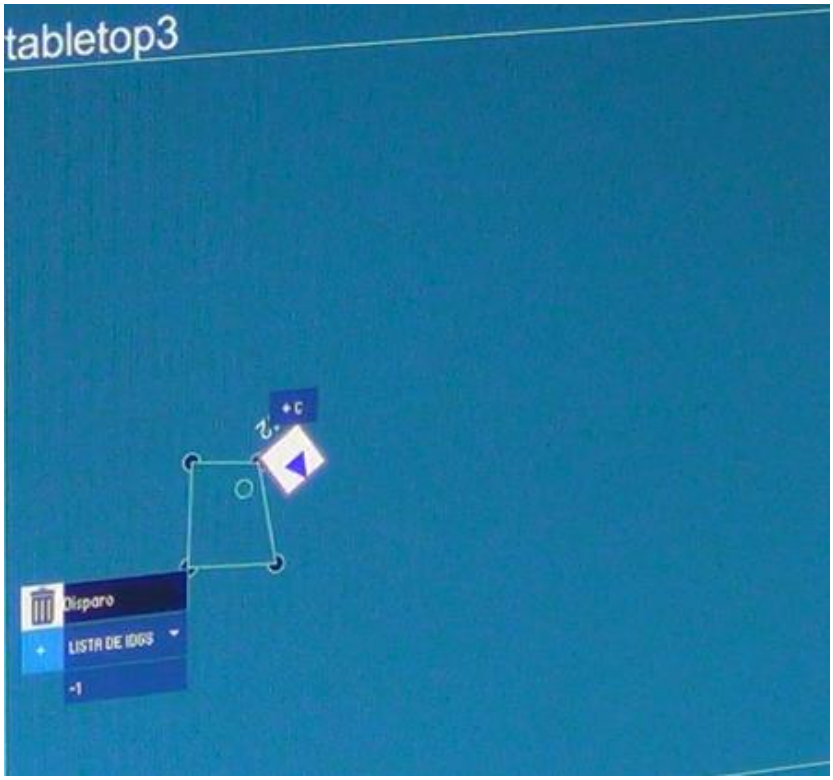


Imagen 34: Imagen ampliada de la pantalla de la creación del constraint "Disparo"

En la imagen 35 se observa el servicio Host recibiendo los mensajes del Broadcaster cuando la nave entra en el constraint izquierdo, y en la imagen 38 los mensajes recibidos cuando la nave entra en el constraint derecho.

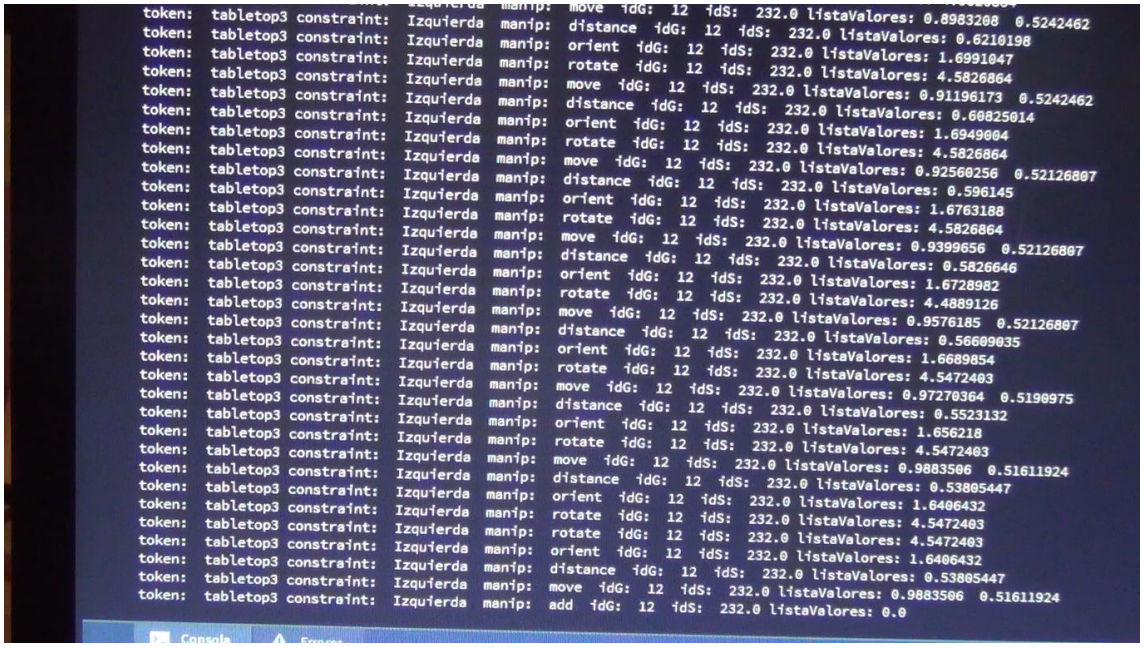


Imagen 35: Mensajes recibidos cuando la nave entra en el constraint "Izquierda"

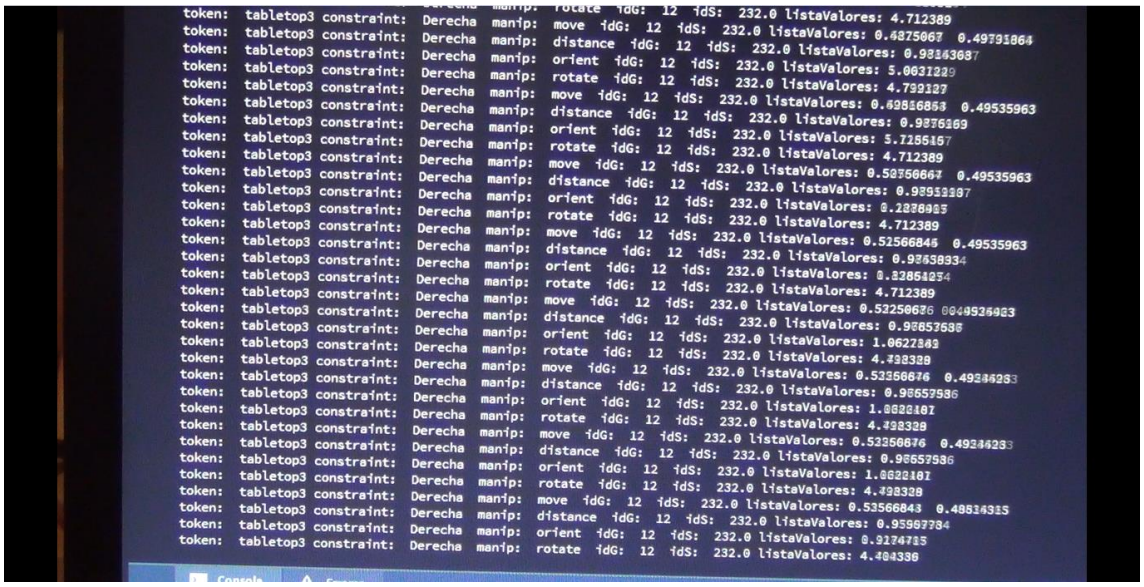


Imagen 36: Mensajes recibidos cuando la nave entra en el constraint "Derecha"

En la imagen 38 se muestran los mensajes recibidos cuando se produce un disparo.

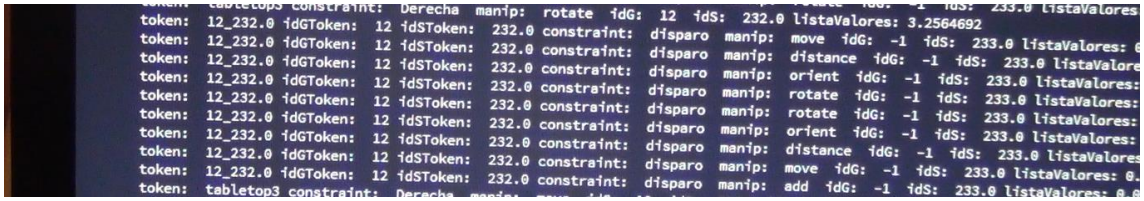


Imagen 37: Imagen ampliada de los mensajes recibidos cuando se dispara con la nave

Como segunda prueba se generó un fichero TUIML para el clásico juego del “Pilla-pilla”, en el que dos jugadores tendrán asociados un localizador con un idG único, gracias al uso de la tecnología RTLS. Se creará un constraint asociado al área de un jugador en el que si el otro jugador entra se enviará un mensaje al Host para simular la idea de que ha sido pillado.

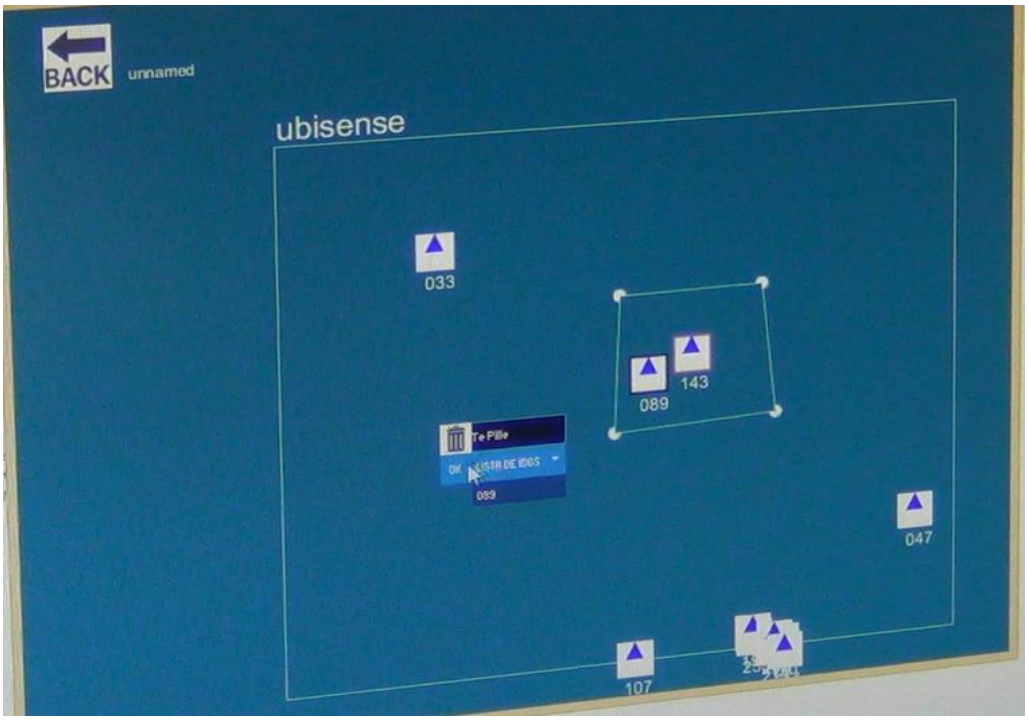


Imagen 38: Imagen ampliada de la creación del constraint "Te Pille"

En la imagen 39 se observa la creación de dicho constraint. Los dos jugadores son el idG 89 y el 143. Se ha creado un constraint asociado al jugador 143, en el que, si entra el jugador 89, se enviará un mensaje al Host de que el 143 ha sido pillado. Los mensajes enviados se pueden observar en la imagen 40.

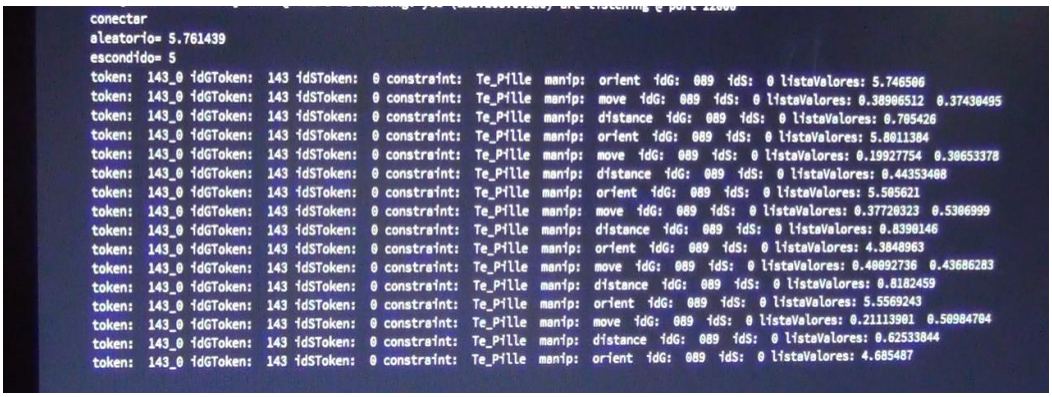


Imagen 39: Imagen ampliada de los mensajes recibidos cuando el token 89 pilla al token 143

Con estas pruebas se ha comprobado el correcto funcionamiento del editor en todos los requisitos planteados inicialmente, y así se puede asegurar que el proyecto funciona en el espacio interactivo.

7 Conclusiones y trabajo futuro

Uno de los principales objetivos de este trabajo era desarrollar una interfaz que facilitase la codificación de un juego pervasivo, para que a diseñadores y desarrolladores no expertos en juegos pervasivos les fuera más fácil el proceso de diseño y desarrollo.

Este trabajo ha testeado la interacción con el usuario, y se ha diseñado e implementado siempre teniendo en cuenta las necesidades del usuario en mente, por lo que se han efectuado las pruebas necesarias.

Como conclusión del trabajo se puede afirmar que:

- Se ha logrado especificar las manipulaciones significativas de un determinado juego pervasivo de forma gráfica usando el lenguaje de modelado TUIML. Ahora, con el uso de la aplicación se puede hacer en tiempo real, modificando el propio TUIML mientras el sistema está en funcionamiento.
- Se ha conseguido mostrar por pantalla los elementos propios de TUIML actualizados según los mensajes OSC que llegaban de los distintos dispositivos.
- Se ha logrado la exportación de lenguaje TUIML de forma correcta, y se ha mejorado la importación de TUIML para soportar una visualización gráfica.
- Se ha probado la aplicación de forma exitosa en el espacio interactivo en Etopia.
- Se han realizado pruebas iterativas sobre el diseño de la interfaz para pulir detalles de usabilidad.

Como trabajo futuro, uno de los objetivos sería poder tratar también con Constraints de tipo 3D, y poder expandir el diseño y alcance de los juegos pervasivos más allá de superficies 2D. Para poder generar estructuras TUIML asociadas a sensores 3D, habría que:

- Modificar el editor de forma que se puedan seleccionar áreas en un espacio 3D, además de también mostrar la posición, movimiento y rotación de un objeto 3D, siendo esto una tarea mucho más compleja que representar objetos 2D.
- Poder exportar e importar ficheros TUIML también se complicaría, ya que los constraints asociados a subtokens tendrían que ser mucho más precisas al tener que calcular en todo momento sus posiciones, y tener que plasmar eso en estructura TUIML.
- Adaptar el editor a gestionar mensajes OSC de sensores 3D conllevaría aumentar las prestaciones de los equipos donde se ejecuta el proceso Broadcaster y tener una tecnología que recogiera posición y movimiento en un plano 3D.

Otro objetivo futuro sería poder generalizar la funcionalidad de la aplicación para que fuera utilizable para otros toolkits que trabajasen con el diseño e implementación de juegos pervasivos, y que no sólo funcione para el toolkit JUGUEMOS. Dicho objetivo sería bastante complejo ya que otros toolkits se basarían en otros modelos de abstracción, ya que TUIML no es el único lenguaje de modelado de interacción física.

Y, por último, como trabajo futuro queda la realización de pruebas de usabilidad sistemáticas con un mayor número de usuarios. Para ello habría que reunir un mayor número de usuarios reales con distintos perfiles, para poder comprobar que la aplicación se adapta y es fácil de utilizar por diversos tipos de usuario: diseñadores, desarrolladores…

Además, quedaría pendiente utilizar métodos de usabilidad sin usuarios como heurísticas, reuniendo a varios desarrolladores de juegos pervasivos e investigadores del proyecto

JUGUEMOS aplicasen las 10 reglas de Nielsen para obtener información sobre la usabilidad del sistema. Y también poder realizar test de usabilidad con un mayor número de usuarios para poder obtener información tanto subjetiva (percepciones y opiniones de los usuarios) como objetiva o medidas de rendimiento (parametrizar y medir las acciones y los comportamientos que se puedan observar), con la finalidad de obtener medidas tangibles sobre el uso del sistema: tiempo en completar diversas tareas, como la importación de un fichero TUIML, el número de veces que seleccionan una opción incorrecta, el número de veces que piden ayuda para continuar o consultan el manual…

Profesionalmente, el desarrollo de este trabajo ha servido para conocer y desarrollar mis aptitudes en un ámbito de la informática que desconocía, como es la computación ubicua y los juegos pervasivos. Al ser un TFG, ha permitido conocer cómo es el desarrollo, análisis y diseño de un proyecto de Software de investigación académica, en el que uno tiene que aprender a enfocar el trabajo y las tareas en relación con el resto de los investigadores. Con relación a cómo se debe enfocar el trabajo y las tareas, se ha aprendido lo importante que es mantener una comunicación habitual con el cliente (en este caso los investigadores del proyecto JUGUEMOS) de un proyecto Software, para entender el alcance y los requisitos de manera más fácil, y más cuando se trata de un ámbito el cual el desarrollador desconoce.

A nivel personal, ha sido un proyecto que me ha servido para entender la magnitud de un proyecto de Software, y la importancia de gestionar bien las fases del trabajo. Además, me ha servido para poder aplicar los conocimientos adquiridos en asignaturas de la rama de Ingeniería de Software,y también de la asignatura de Interacción Persona-Ordenador, para entender los factores humanos y psicológicos que hay detrás de la interacción con el Software.

8 Referencias

[W93] Weiser, M. (1993). Hot topics-ubiquitous computing. Computer, *26*(10), 71-72.

[MCMN05] Magerkurth, C., Cheok, A. D., Mandryk, R. L., & Nilsen, T. (2005). Pervasive games: bringing computer entertainment back to the real world. *Computers in Entertainment (CIE), 3*(3), 4-4.

[HLM07] Hinske, S., Lampe, M., Magerkurth, C., & Röcker, C. (2007). Classifying pervasive games: on pervasive computing and mixed reality. *Concepts and technologies for Pervasive Games-A Reader for Pervasive Gaming Research*, *1*(20).

[IU97] Ishii, H., & Ullmer, B. (1997, March). Tangible bits: towards seamless interfaces between people, bits and atoms. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems* (pp. 234-241). ACM.

[SJ09] Shaer, O., & Jacob, R. J. (2009). A specification paradigm for the design and implementation of tangible user interfaces. *ACM Transactions on Computer-Human Interaction (TOCHI)*, *16*(4), 20.

[MCB12] Marco, J., Cerezo, E., Baldassarri, S. Tangible Interaction and Tabletops: New Horizons for Children's Games. International Journal of Arts and Technology (IJART). Vol. 5, Nos. 2/3/4. 2012. pp.151-176 ISSN: 1754-8853. Ed. Inderscience.

[SLC04] Shaer, O., Leland, N., Calvillo-Gamez, E. H., & Jacob, R. J. (2004). The TAC paradigm: specifying tangible user interfaces. *Personal and Ubiquitous Computing*, *8*(5), 359-369.

[CMB15] Cerezo, E., Marco, J., & Baldassarri, S. (2015). Hybrid games: designing tangible interfaces for very young children and children with special needs. In *More Playful User Interfaces* (pp. 17-48). Springer, Singapore.

[DG02] Davies, N., & Gellersen, H. W. (2002). Beyond prototypes: Challenges in deploying ubiquitous systems. *IEEE Pervasive computing*, *1*(1), 26-35.

[MEG06] Magerkurth, C., Engelke, T., & Grollman, D. (2006, June). A component-based architecture for distributed, pervasive gaming applications. In *Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology* (p. 15). ACM.

[MCB12] Marco, J., Cerezo, E., & Baldassarri, S. (2012, June). ToyVision: a toolkit for prototyping tabletop tangible games. In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems* (pp. 71-80). ACM.

[MCB08] Marco, J., Cerezo, E., & Baldassarri, S. (2008). NikVision: Desarrollo de Videojuegos basados en Interfaces Naturales. In *CEIG* (pp. 233-236).

[BR00] Boren, T., & Ramey, J. (2000). Thinking aloud: Reconciling theory and practice. *IEEE transactions on professional communication*, *43*(3), 261-278

[BID99] Bobick, A. F., Intille, S. S., Davis, J. W., Baird, F., Pinhanez, C. S., Campbell, L. W., ... & Wilson, A. (1999). The KidsRoom: A perceptually-based interactive and immersive story environment. *Presence*, *8*(4), 369-393.

[GLC15] PFC de Guillermo Lafuente Gallego, "*Desarrollo de un editor gráfico basado en un lenguaje de modelado visual para juegos híbridos"* https://zaguan.unizar.es/record/37100?ln=es

[TOY] Página web con referencias a ToyVision: https://zaguan.unizar.es/record/64190?ln=es

[CP5] Página web de la librería ControlP5: http://www.sojamo.de/libraries/controlP5/

[OSC] Página web de la librería oscP5: http://www.sojamo.de/libraries/oscP5/

[PRO] Página Web de Processing: https://processing.org/

[W05] Wright, M. (2005). Open Sound Control: an enabling technology for musical networking. *Organised Sound*, *10*(3), 193-200

[REA] Página web del software ReacTIVision https://github.com/mkalten/reacTIVision

[G4P] Página web de la librería G4P: http://www.lagers.org.uk/g4p/

[GIG14] Giga Affective Lab: http://giga.cps.unizar.es/affectivelab/

9 Anexos

9.1 Anexo A: Diagramas de secuencia

En este anexo se adjuntan los diagramas de secuencia donde se especifica la interacción entre las distintas partes del sistema y el usuario.

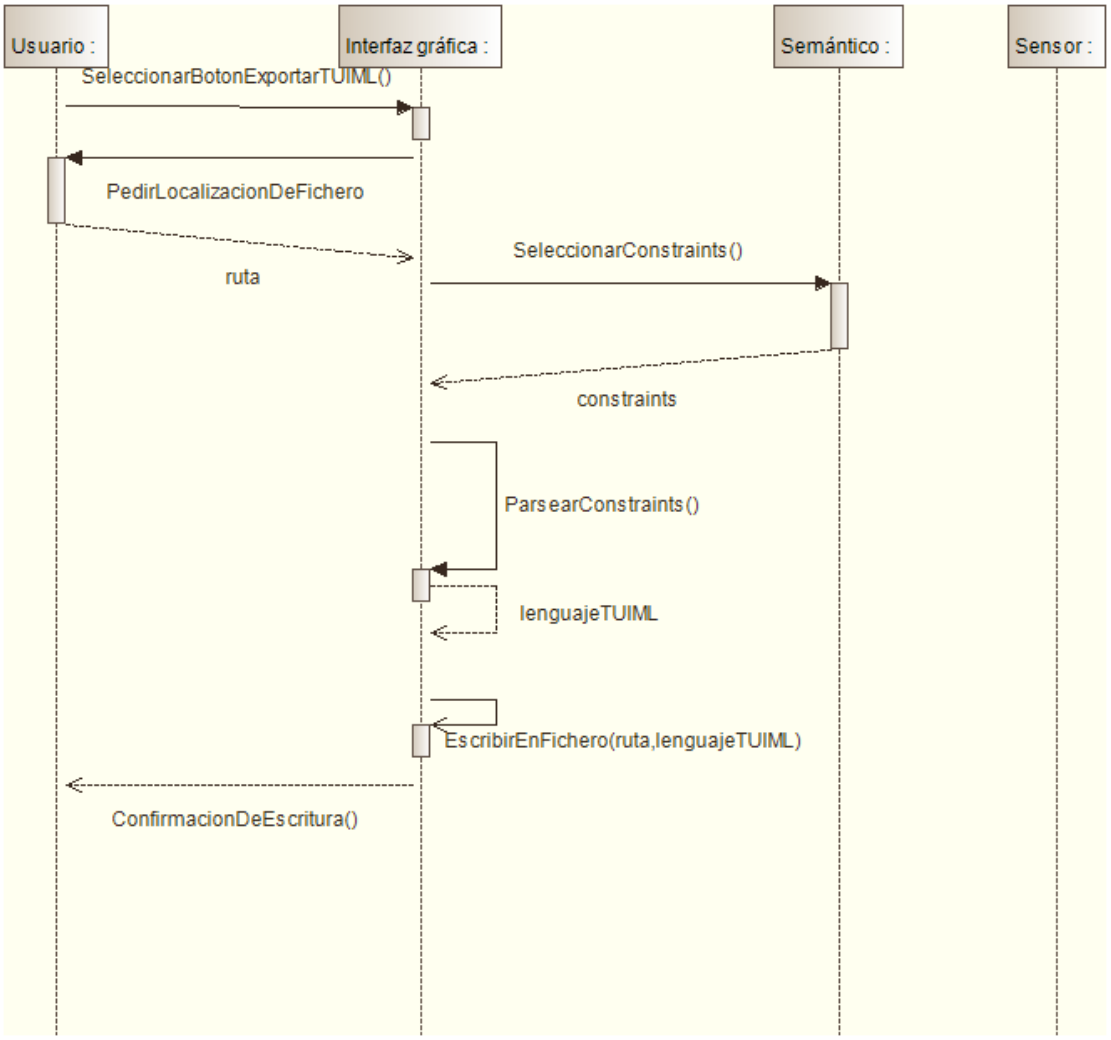


Imagen 40:Diagrama de secuencia del caso de uso "Exportar TUIML"

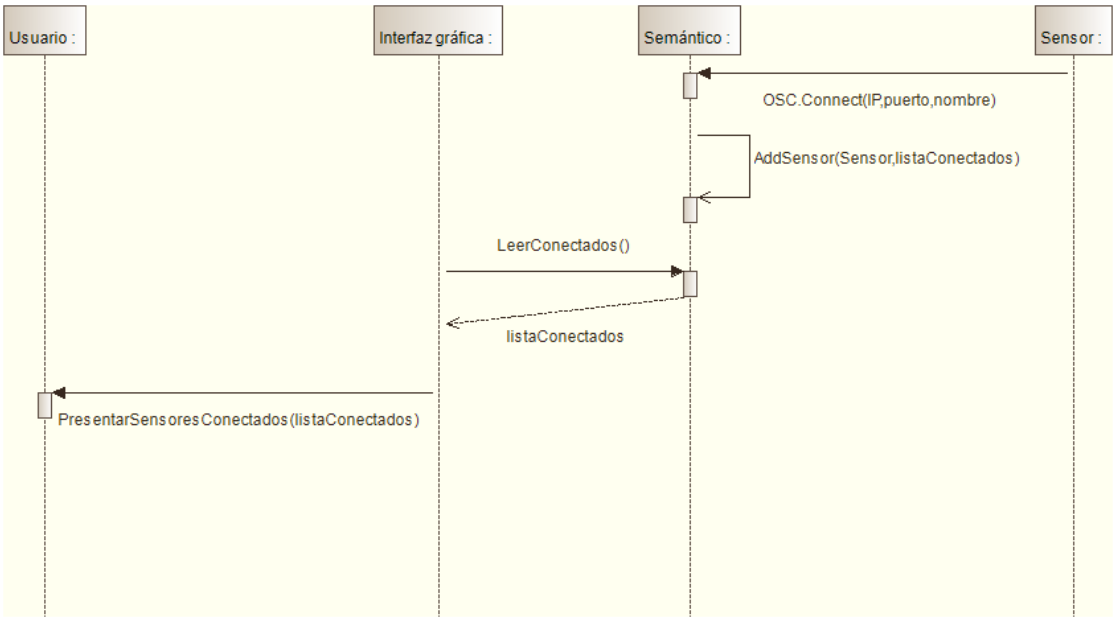


Imagen 41:Diagrama de secuencia de los casos de uso "Visualización de la información de sensores 0D,1D,2D"

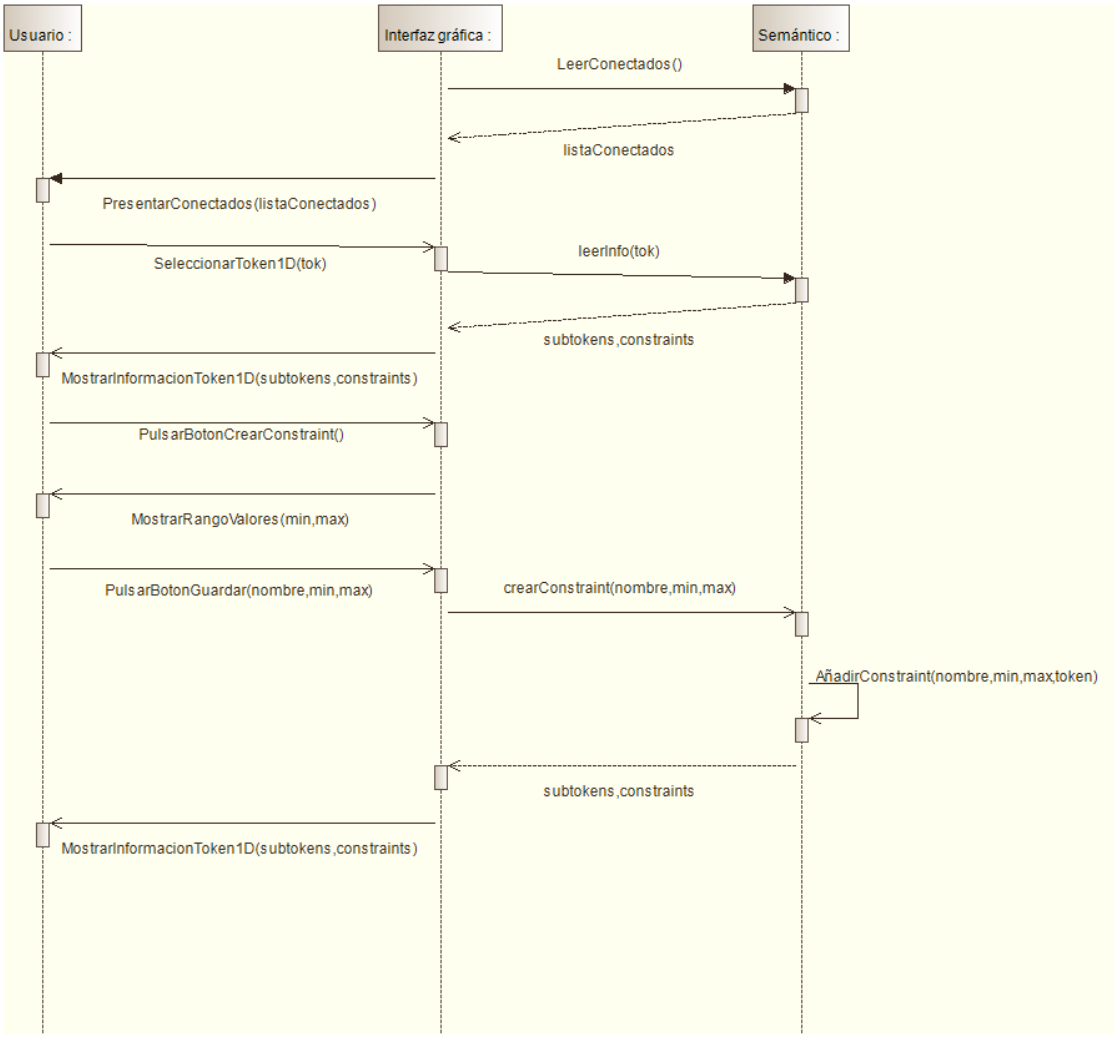


Imagen 42: Diagrama de secuencia del caso de uso "Crear constraint 1D"

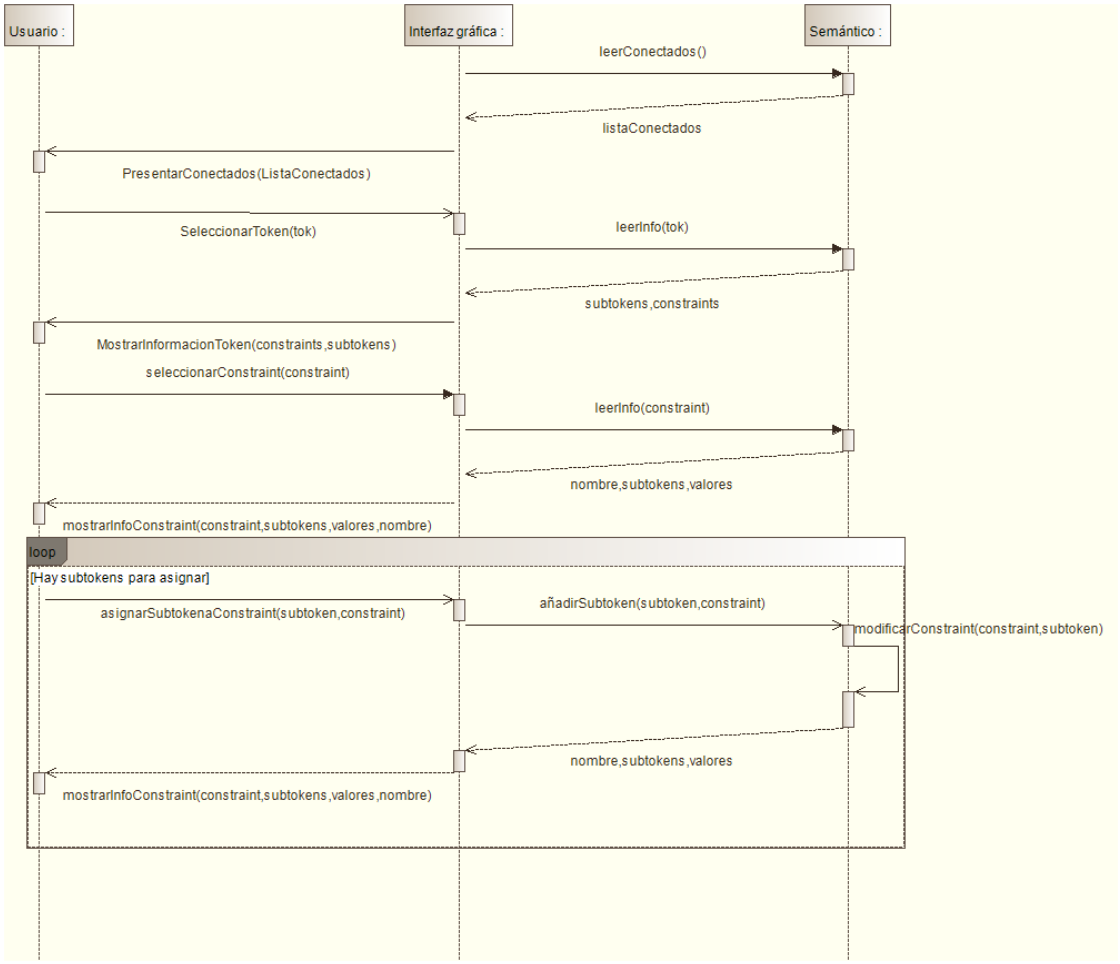


Imagen 43: Diagrama de secuencia de los casos de uso "Asignar subtoken a constraint 1D y 2D"

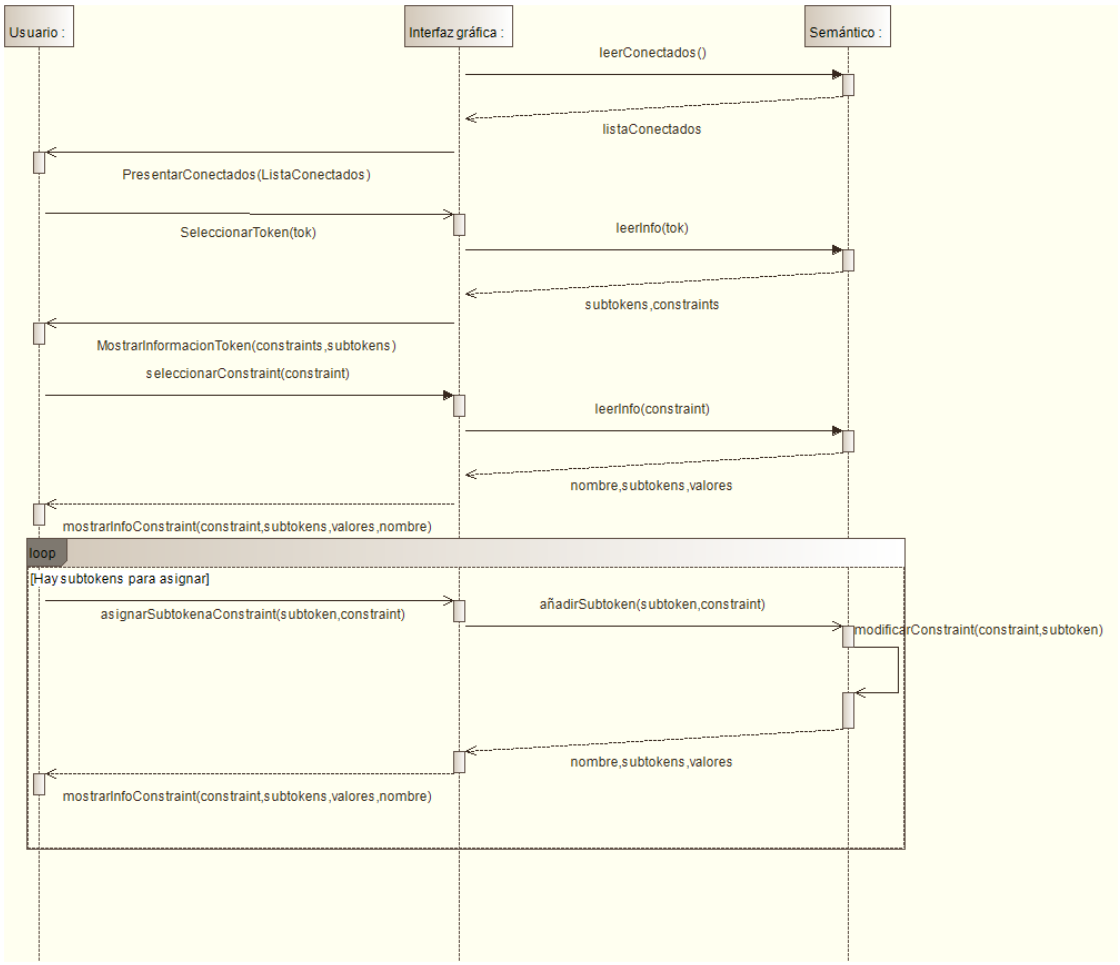


Imagen 44: Diagrama de secuencia de los casos de uso "Modificar constraints 2D y 1D", en el que se modifica el nombre de un constraint existente

Imagen 45:Diagrama de secuencia que muestra la interacción entre el sistema y el usuario para mostrar los constraints 2D

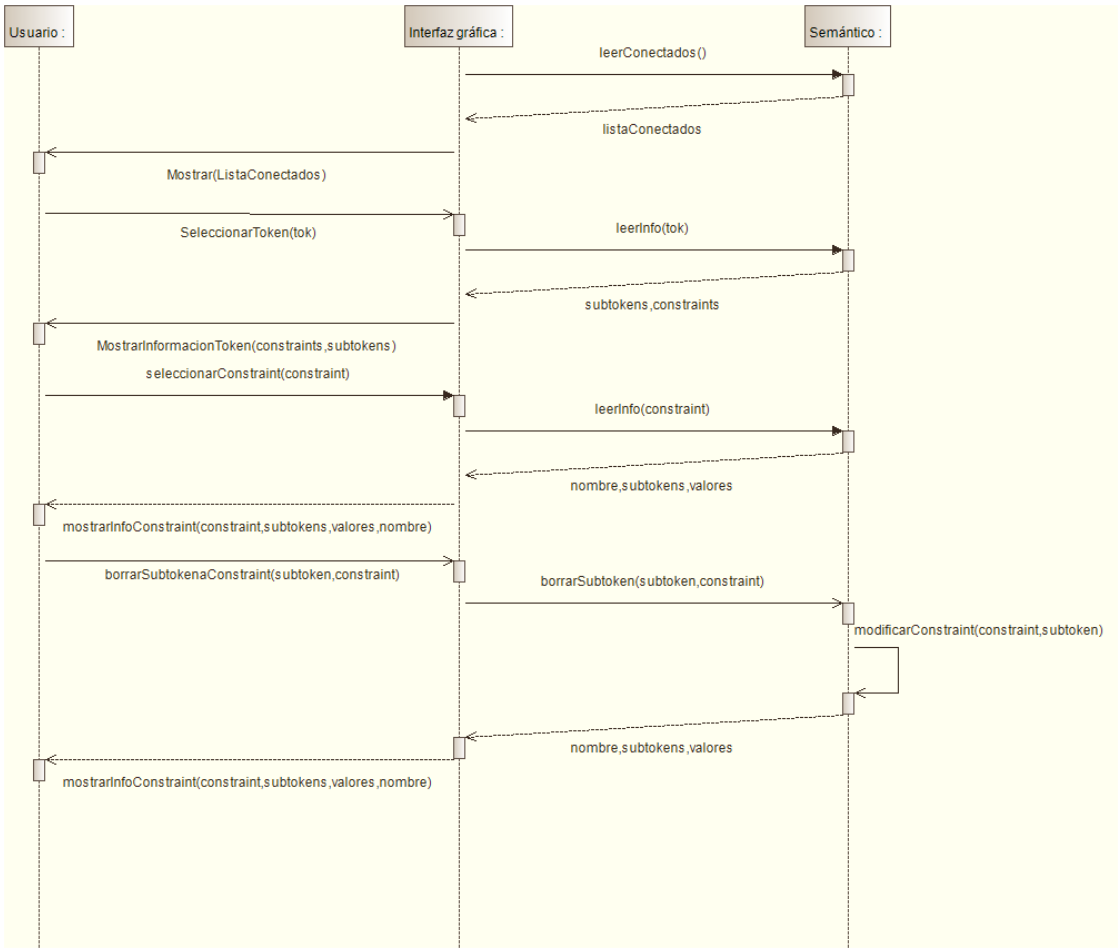


Imagen 46: Diagrama de secuencia que representa el caso de uso "Eliminar subtokens a constraints 1D y 2D"

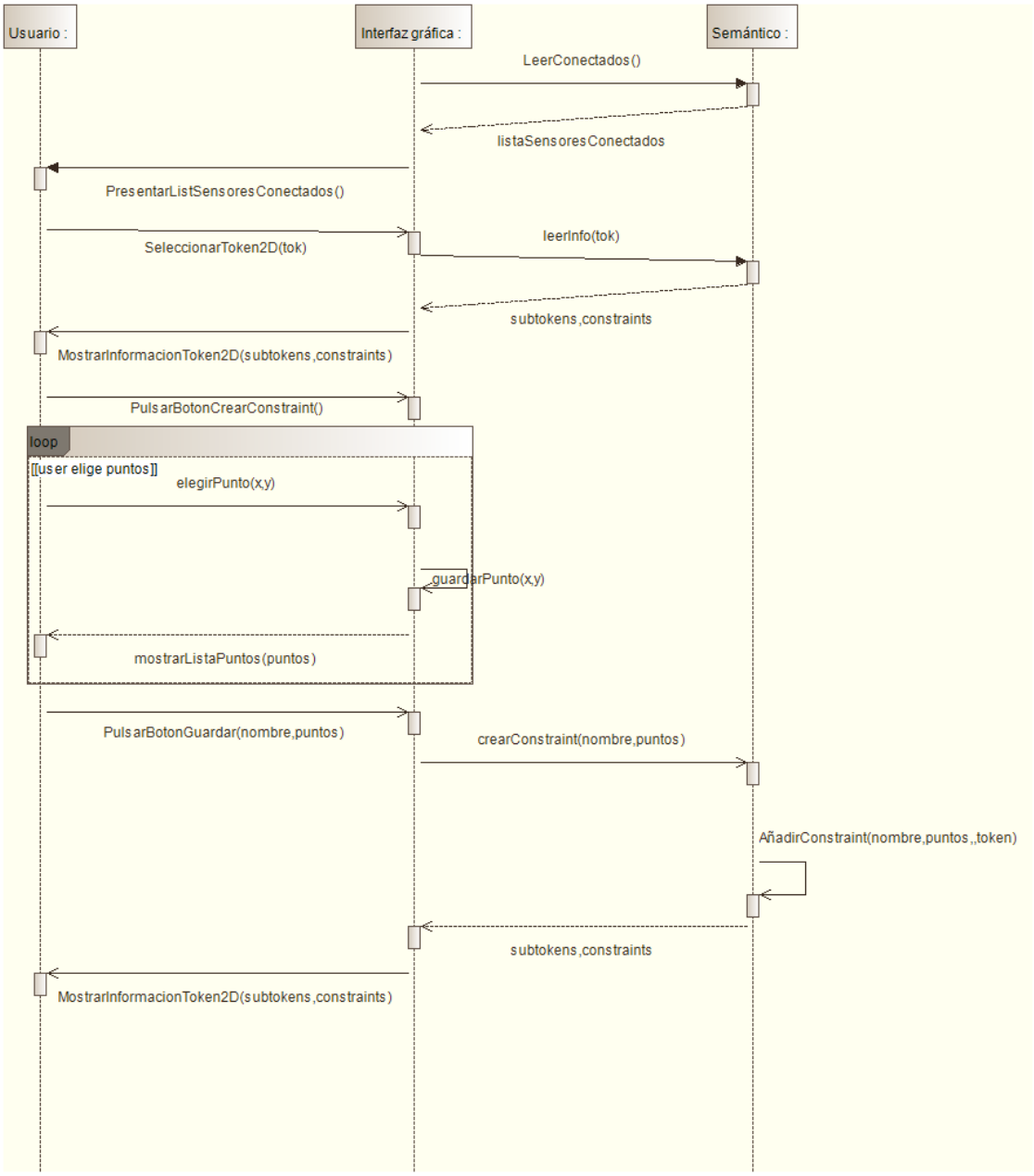


Imagen 47: Diagrama de secuencia que representa el caso de uso "crear constraint 2D"

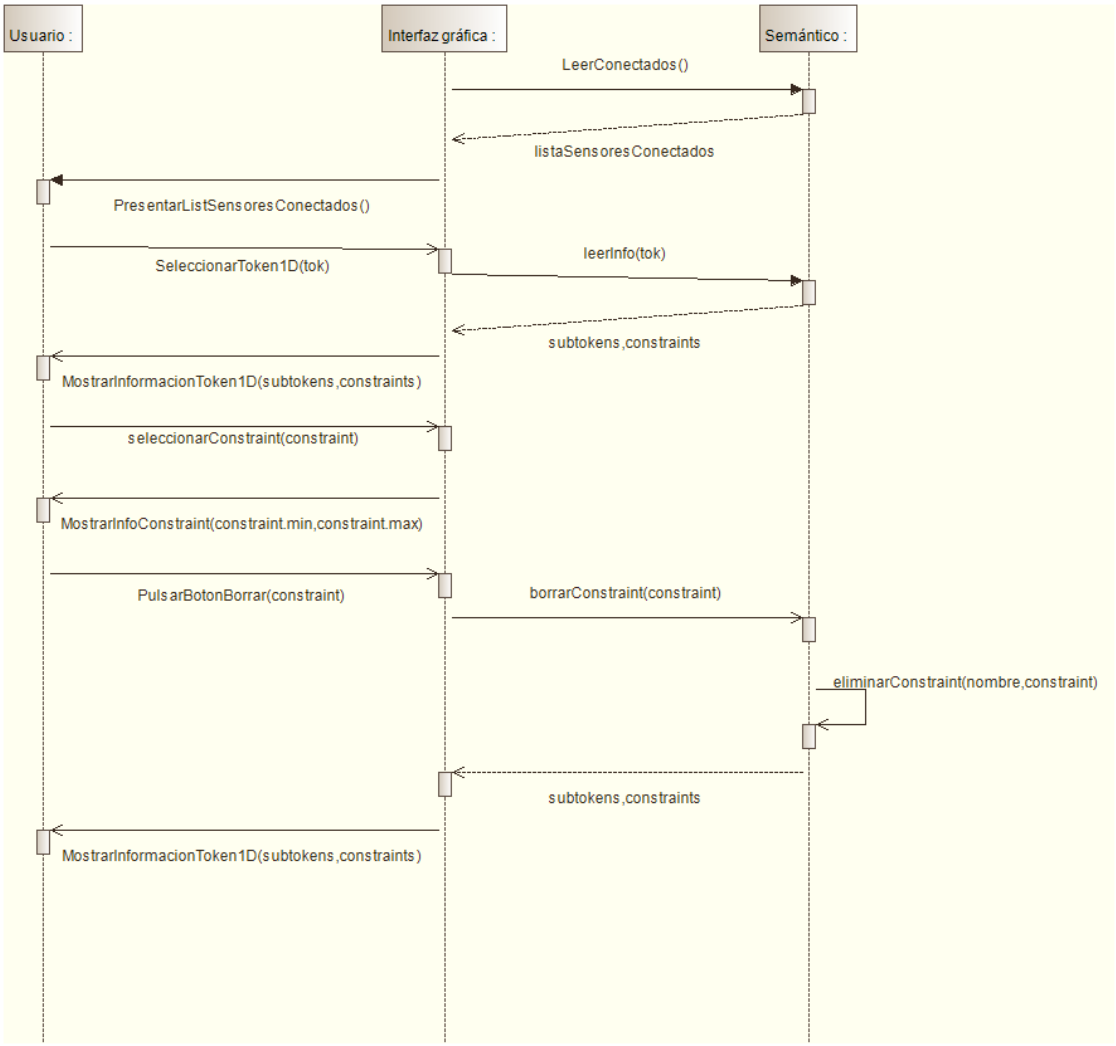


Imagen 48: Diagrama de secuencia que representa el caso de uso "Borrar constraint 1D"

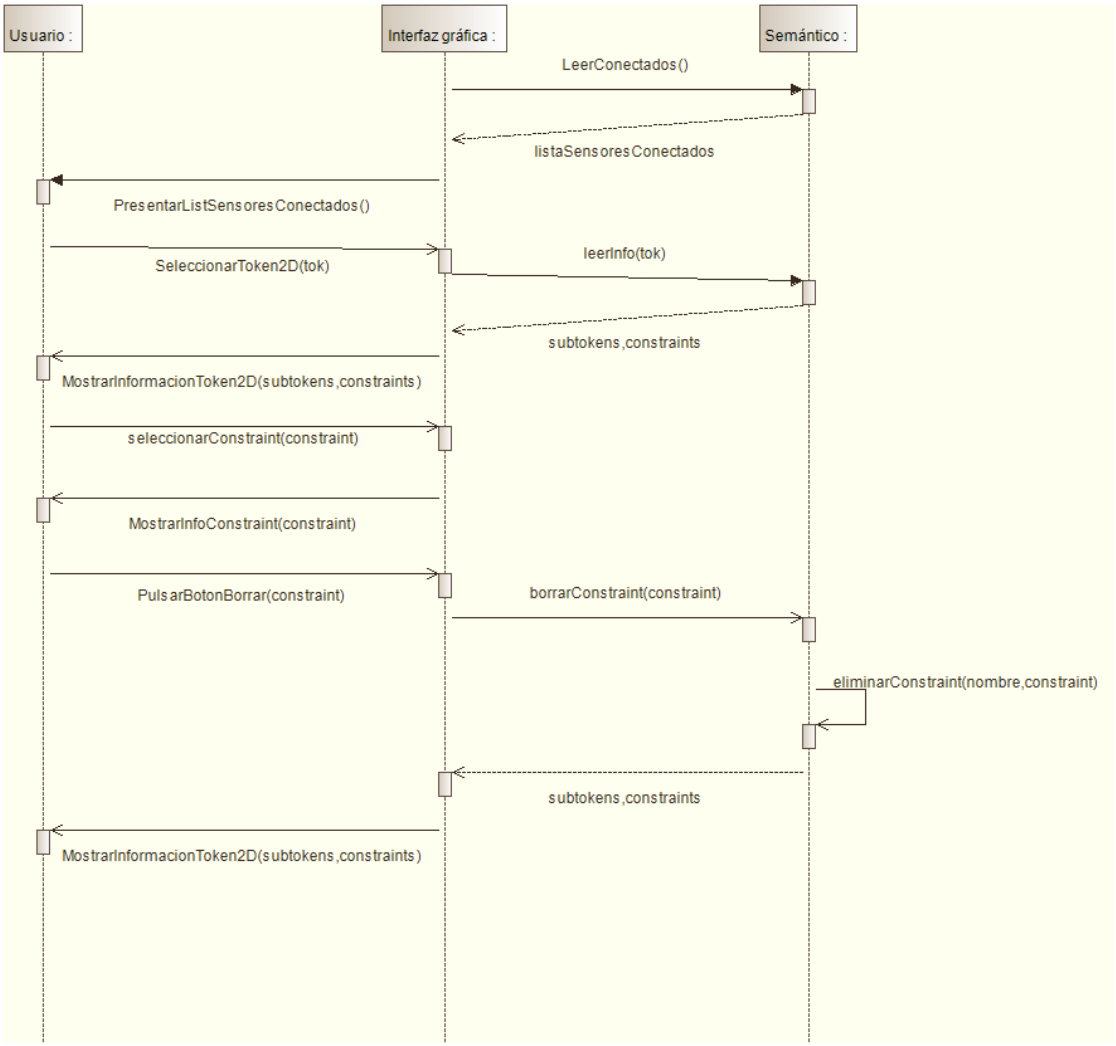


Imagen 49: Diagrama de secuencia que representa el caso de uso "Borrar constraint 2D"

9.2 Anexo B: Casos de uso especificados

ID:	01
Nombre:	Visualizar información de sensor 0D

Actor:	Usuario
Descripción:	El usuario visualiza los datos recibidos por un sensor de tipo 0D
Precondiciones:	--
Postcondiciones:	--
Flujo normal de eventos:	1. El sensor 0D inicia conexión con el editor gráfico 2. El editor gráfico añade el sensor a su lista de sensores conectados 3. El usuario selecciona el sensor 0D 4. El editor gráfico muestra la pantalla con la información recibida.
Flujo alternativo:	--

ID:	02

Nombre:	Visualizar información de sensor 1D

Actor:	Usuario
Descripción:	El usuario visualiza los datos recibidos por un sensor de tipo 1D
Precondiciones:	--
Postcondiciones:	--
Flujo normal de eventos:	1. El sensor 1D inicia conexión con el editor gráfico 2. El editor gráfico añade el sensor a su lista de sensores conectados 3. El usuario selecciona el sensor 1D 4. El editor gráfico muestra la pantalla con la información recibida.
Flujo alternativo:	--

ID:	03
Nombre:	Visualizar información de sensor 2D

Actor:	Usuario
Descripción:	El usuario visualiza los datos recibidos por un sensor de tipo 2D
Precondiciones:	--
Postcondiciones:	--
Flujo normal de eventos:	1. El sensor 2D inicia conexión con el editor gráfico 2. El editor gráfico añade el sensor a su lista de sensores conectados 3. El usuario selecciona el sensor 2D 4. El editor gráfico muestra la pantalla con la información recibida.
Flujo alternativo:	--

ID:	04
Nombre:	Marcar token 0D

Actor:	Usuario
Descripción:	El usuario selecciona un token 0D y lo marca
Precondiciones:	--
Postcondiciones:	--
Flujo normal de eventos:	1. <<include>> *Visualizar sensor 0D* 2. El editor gráfico le presenta al usuario todos los valores 0D que ha recibido del sensor 3. El usuario selecciona uno de ellos 4. El editor gráfico muestra la elección del usuario en la interfaz, mediante el cambio en el borde del elemento.
Flujo alternativo:	--

ID:	05
Nombre:	Desmrcar token 0D

Actor:	Usuario
Descripción:	El usuario selecciona un token 0D marcado y lo desmarca

Precondiciones:	El elemento que el usuario desmarca debe estar marcado de antemano.
Postcondiciones:	--
Flujo normal de eventos:	1. <<include>> *Visualizar sensor 0D* 2. El editor gráfico le presenta al usuario todos los valores 0D que ha recibido del sensor 3. El usuario selecciona uno de ellos 4. El editor gráfico muestra la elección del usuario en la interfaz, mediante el cambio en el borde del elemento.
Flujo alternativo:	--

ID:	06
Nombre:	Añadir subtoken para constraints 1D

Actor:	Usuario
Descripción:	El usuario selecciona un token a quien adjuntar el constraint que se va a crear.
Precondiciones:	El usuario debe haber elegido visualizar un sensor 1D.
Postcondiciones:	Cuando se cree un nuevo constraint, se asignará junto al subtoken elegido.
Flujo normal de eventos:	1. El editor gráfico le presenta al usuario todos los valores 1D que ha recibido del sensor 2. El usuario selecciona un token dentro del área donde se representan los elementos 1D.
Flujo alternativo:	--

ID:	07
Nombre:	Añadir subtoken para constraints 2D

Actor:	Usuario
Descripción:	El usuario selecciona un token a quien adjuntar el constraint que se va a crear.
Precondiciones:	El usuario debe haber elegido visualizar un sensor 2D.
Postcondiciones:	Cuando se cree un nuevo constraint, se asignará junto al subtoken elegido.
Flujo normal de eventos:	1. El editor gráfico le presenta al usuario todos los valores 2D que ha recibido del sensor 2. El usuario selecciona un token dentro del área donde se representan los elementos 2D.
Flujo alternativo:	--

ID:	08
Nombre:	Eliminar subtoken del constraint 2D

Actor:	Usuario
Descripción:	El usuario elimina de la lista de subtokens relacionados con cierto constraint, el subtoken deseado
Precondiciones:	La lista de subtokens de un constraint debe ser mayor que 0.

Postcondiciones:	--
Flujo normal de eventos:	1. El editor gráfico le presenta al usuario todos los subtokens 2D relacionados con el constraint seleccionado. 2. El usuario selecciona el subtoken a borrar 3. El editor gráfico actualiza la lista
Flujo alternativo:	--

ID:	09
Nombre:	Eliminar subtoken del constraint 1D

Actor:	Usuario
Descripción:	El usuario elimina de la lista de subtokens relacionados con cierto constraint, el subtoken deseado
Precondiciones:	La lista de subtokens de un constraint debe ser mayor que 0.
Postcondiciones:	--
Flujo normal de eventos:	1. El editor gráfico le presenta al usuario todos los subtokens 1D relacionados con el constraint seleccionado. 2. El usuario selecciona el subtoken a borrar 3. El editor gráfico actualiza la lista
Flujo alternativo:	--

ID:	10
Nombre:	Crear constraint 1D

Actor:	Usuario
Descripción:	El usuario crea un nuevo constraint, relacionado con un token 1D.
Precondiciones:	--
Postcondiciones:	Se añade un constraint a la lista de constraints del token 1D
Flujo normal de eventos:	1. <<include>> *Visualizar información de sensor 1D* 2. El usuario selecciona el botón de "*crear constraint*". 3. El usuario introduce el nombre del constraint. 4. El usuario selecciona el valor inferior del constraint dentro de un rango de 0 a 1. 5. El usuario selecciona el valor superior del constraint dentro de un rango de 0 a 1. 6. El usuario selecciona el botón de aceptar. 7. El editor gráfico actualiza la lista de constraints.
Flujo alternativo:	• Si <<extend>> *Añadir subtoken 1D a un constraint*: El constraint a crear estará relacionado con dicho subtoken. • Si el nombre introducido por el usuario ya existe: El editor gráfico cancela la operación y avisa al usuario del error.

ID:	11
Nombre:	Crear constraint 2D

Actor:	Usuario
Descripción:	El usuario crea un nuevo constraint, relacionado con un token 2D.
Precondiciones:	--
Postcondiciones:	Se añade un constraint a la lista de constraints del token 2D

Flujo normal de eventos:	1. <<include>> *Visualizar información de sensor 2D* 2. El usuario selecciona el botón de "*crear constraint*". 3. El usuario introduce el nombre del constraint. 4. El usuario selecciona diversos puntos en el editor gráfico hasta crear el área del constraint. 5. El usuario selecciona el botón de aceptar. 6. El editor gráfico actualiza la lista de constraints.
Flujo alternativo:	• Si <<extend>> *Añadir subtoken 2D a un constraint*: El constraint a crear estará relacionado con dicho subtoken. • Si el nombre introducido por el usuario ya existe: El editor gráfico cancela la operación y avisa al usuario del error.

ID:	12
Nombre:	Eliminar constraint 2D

Actor:	Usuario
Descripción:	El usuario elimina un constraint, relacionado con un token 2D.
Precondiciones:	--
Postcondiciones:	--
Flujo normal de eventos:	1. <<include>> *Visualizar información de sensor 2D* 2. El usuario selecciona un constraint de la lista de constraints. 3. El editor gráfico eleimina dicho constraint de la lista y la actualiza
Flujo alternativo:	--

ID:	13
Nombre:	Eliminar constraint 1D

Actor:	Usuario
Descripción:	El usuario elimina un constraint, relacionado con un token 1D.
Precondiciones:	--
Postcondiciones:	--
Flujo normal de eventos:	1. <<include>> *Visualizar información de sensor 1D* 2. El usuario selecciona un constraint de la lista de constraints. 3. El editor gráfico eleimina dicho constraint de la lista y la actualiza
Flujo alternativo:	--

ID:	14
Nombre:	Modificar constraint 1D

Actor:	Usuario
Descripción:	El usuario modifica un constraint ya existente, relacionado con un token 1D.

Precondiciones:	--
Postcondiciones:	--
Flujo normal de eventos:	1. <<include>> *Visualizar información de sensor 1D* 2. El usuario selecciona un constraint de la lista de constraints. 3. El usuario modifica los campos del constraint: nombre, valores. 4. El editor gráfico actualiza el constraint
Flujo alternativo:	• Si <<extend>> *Eliminar subtoken del constraint 1D:* El usuario habrá eliminado un subtoken de la lista de subtokens del constraint.

ID:	15
Nombre:	Modificar constraint 2D

Actor:	Usuario
Descripción:	El usuario modifica un constraint ya existente, relacionado con un token 2D.
Precondiciones:	--
Postcondiciones:	--
Flujo normal de eventos:	1. <<include>> *Visualizar información de sensor 2D* 2. El usuario selecciona un constraint de la lista de constraints. 3. El usuario modifica los campos del constraint: nombre, valores. 4. El editor gráfico actualiza el constraint
Flujo alternativo:	• Si <<extend>> *Eliminar subtoken del constraint* 2D*:* El usuario habrá eliminado un subtoken de la lista de subtokens del constraint.

ID:	16
Nombre:	Exportar TUIML

Actor:	Usuario
Descripción:	El usuario exporta los constraints creados a lenguaje TUIML
Precondiciones:	--
Postcondiciones:	--
Flujo normal de eventos:	1. El usuario selecciona el fichero donde escribir el lenguaje TUIML 2. El sistema escribe el lenguaje TUIML en el fichero.
Flujo alternativo:	• Si el usuario escribe el nombre de un fichero que no existe, el sistema lo creará.

ID:	17
Nombre:	Importar TUIML

Actor:	Usuario
Descripción:	El usuario importa los constraints y subtokens de un fichero en lenguaje TUIML y los muestra de forma gráfica.
Precondiciones:	--

Postcondiciones:	--
Flujo normal de eventos:	1. El usuario selecciona el fichero con lenguaje TUIML. 2. El sistema parsea y lee los constraints del fichero. 3. El sistema muestra lo extraído del fichero en pantalla.
Flujo alternativo:	• Si el usuario selecciona un fichero con sintaxis incorrecta, el sistema informa del error y aborta la importación.

9.3 Anexo C: Ejemplo de fichero TUIML.

```
<juguemos>
    <token name="tabletop1">
        <constraint name="todo" type= "2D" list_vertex="0.0,0.0,1.0,0.0,1.0,1.0,0.0,1.0,0.0,0.0">
            <tac subtoken="12,15"/>
        </constraint>
        <subtoken idG="12">
            <constraint name="disparo" type= "2D" list_vertex="
            0.029807977,0.07958425,0.029807992,-0.05868735,0.08814133,-0.05127991,0.08536353,0.08699168,0.031196885,0.077115126">
                <tac subsubtoken="-1"/>
            </constraint>
            <subsubtoken idG ="-1"/>
        </subtoken>
        <subtoken idG="15">
            <constraint name="disparo" type= "2D" list_vertex="
            0.029807977,0.07958425,0.029807992,-0.05868735,0.08814133,-0.05127991,0.08536353,0.08699168,0.031196885,0.077115126">
                <tac subsubtoken="-1"/>
            </constraint>
            <subsubtoken idG ="-1"/>
        </subtoken>
    </token>
<token name="tabletop2">
        <constraint name="todo" type= "2D" list_vertex="0.0,0.0,1.0,0.0,1.0,1.0,0.0,1.0,0.0,0.0">
            <tac subtoken="12,15"/>
        </constraint>
        <subtoken idG="12">
            <constraint name="disparo" type= "2D" list_vertex="
            0.029807977,0.07958425,0.029807992,-0.05868735,0.08814133,-0.05127991,0.08536353,0.08699168,0.031196885,0.077115126">
                <tac subsubtoken="-1"/>
            </constraint>
            <subsubtoken idG ="-1"/>
        </subtoken>
        <subtoken idG="15">
            <constraint name="disparo" type= "2D" list_vertex="
            0.029807977,0.07958425,0.029807992,-0.05868735,0.08814133,-0.05127991,0.08536353,0.08699168,0.031196885,0.077115126">
                <tac subsubtoken="-1"/>
            </constraint>
            <subsubtoken idG ="-1"/>
        </subtoken>
    </token>
```

Imagen 50: Ejemplo de un juego creado para el espacio, conocido como "Asteroides"

En la imagen 50 se observa un pantallazo de un fichero *TUIML* de un juego en específico.

En este juego podemos observar los siguientes *tokens* y sus relaciones:

- Token *tabletop1*: Este token 2D representa una de las mesas interactivas disponibles en el laboratorio en Etopia, y dicha mesa tiene un *constraint* que se ha nombrado como *todo*. Dicho constraint ocupa un área representada por los pares de puntos en *list_vertex* y mantiene una relación con los *subtokens* 12 y 15, esto es que cuando uno de los dos *subtokens* (o ambos) con estos identificadores se introduzcan en el área determinada por el constraint *todo*, generará algún tipo de interacción definida por el desarrollador.
- Subtoken *12*: Este subtoken 2D representa un objeto físico (en este caso un juguete) identificado con el número 12. Como se observa, dicho *subtoken* también tiene un *constraint*, conocido como *disparo* y delimitado por los pares de coordenadas de *list_vertex*, además de mantener una relación con el *subsubtoken* identificado por -1.
- El token *tabletop2* mantiene las mismas relaciones que el token *tabletop1*.

Con este ejemplo se pretende ilustrar la naturaleza estructurada de los ficheros TUIML del espacio interactivo.

9.4 Anexo D: Seguimiento temporal

En este anexo se muestra la distribución del tiempo empleado en la realización de este proyecto.

En la imagen 51 se muestra el diagrama de Gantt del proyecto, y en la imagen 52 se muestra su organización de tareas.

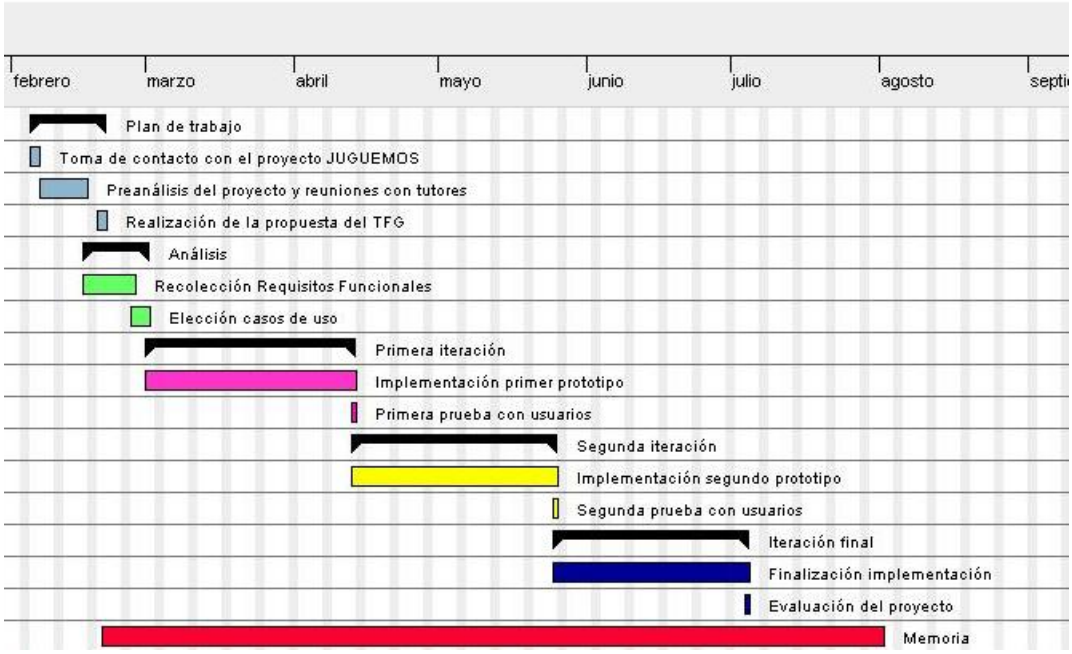


Imagen 51: Diagrama de Gantt del proyecto

Nombre	Fecha de inicio	Fecha de fin
Plan de trabajo	5/02/18	20/02/18
Toma de contacto con el proyecto JUGUEMOS	5/02/18	6/02/18
Preanálisis del proyecto y reuniones con tutores	7/02/18	16/02/18
Realización de la propuesta del TFG	19/02/18	20/02/18
Análisis	16/02/18	1/03/18
Recolección Requisitos Funcionales	16/02/18	26/02/18
Elección casos de uso	26/02/18	1/03/18
Primera iteración	1/03/18	13/04/18
Implementación primer prototipo	1/03/18	13/04/18
Primera prueba con usuarios	13/04/18	13/04/18
Segunda iteración	13/04/18	25/05/18
Implementación segundo prototipo	13/04/18	25/05/18
Segunda prueba con usuarios	25/05/18	25/05/18
Iteración final	25/05/18	4/07/18
Finalización implementación	25/05/18	4/07/18
Evaluación del proyecto	4/07/18	4/07/18
Memoria	20/02/18	1/08/18

Imagen 52: Organización en tareas del proyecto

9.5 Anexo E: Explicación del diagrama de clases

En este anexo se va a proceder a explicar de forma más detallada las clases del diagrama de clases (Imagen 28), y las relaciones entre ellas.

9.5.1 Clases Vista

Las vistas son las clases que incluyes todos los elementos gráficos de la librería ControlP5 y todos los procedimientos necesarios para mostrarlos en pantalla.

Hay tres clases de Vista, según el tipo de datos que recibe del sensor 0D,1D, y 2D. Todas ellas extienden a una interfaz, *VistaInterface*, la cual incluye dos procedimientos comunes a todas las clases que lo extiendan, el método *draw()*, para dibujar los elementos por pantalla, y el método *hide()*, para borrar los elementos de la pantalla. A continuación, se comentará cada una de estas tres clases:

- Vista 0D: Incluye los elementos gráficos y procedimientos para los datos de tipo 0D:
    - Atributos de la clase:
        - yCanvas0D: Coordenada x en la que empezar a dibujar el marco.
        - xCanvas0D: Coordenada y en la que empezar a dibujar el marco.
    - Funciones de la clase:
        - PintarSubtokens0D: Recibe una lista de subtokens y las muestra por pantalla, modificando su sombreado según si el subtoken está asociado al constraint o no.
- Vista 1D: Incluye los elementos gráficos y procedimientos para los datos de tipo 1D:
    - Atributos de la clase:
        - micro: Slider con el que se muestra los valores recogidos por el sensor
        - range: Range con el que el usuario selecciona el rango de valores del constraint a crear.
        - nombre1D: Campo de texto con el nombre del constraint.
        - borrar: Botón con el que el usuario borra un constraint
        - arriba: Botón con el que el usuario sube hacia arriba en la lista de constraints.
        - abajo: Botón con el que el usuario sube hacia abajo en la lista de constraints.
        - listaRangos: Lista de rangos que representan los constraints 1D.
        - listaBotones: Lista de botones que representan los subtokens conectados.
        - xInicioRangos: Coordenada x en la que se empieza a mostrar los rangos
        - yInicioRangos: Coordenada y en la que se empieza a mostrar la lista de rangos
        - IndiceRangos: Entero que representa el índice a partir del cual se muestran los rangos de la lista.
    - Procedimientos de la clase:
        - mostrarRangos: Recibe dos enteros que indican que rangos de la lista de rangos se muestran por pantalla.
        - cambiarColorBotones: Modifica el color de los botones según si el constraint seleccionado está asignado al subtoken que representa dicho botón.
- Vista 2D: Incluye los elementos gráficos y procedimientos para los datos de tipo 2D:

    - Atributos de la clase:
        - xCanvas2D: Coordenada x en la que empezar a dibujar el marco.
        - yCanvas2D: Coordenada y en la que empezar a dibujar el marco.
    - Procedimientos de la clase:
        - dibujarConstraintsSubtoken: Dibuja los constraints asociados a un subtoken.
        - dibujarConstraints: Dibuja los constraints asociados al token.
        - dibujarObjetos2D: Se dibuja únicamente el constraint que se le envía como parámetro, y marca con un sombreado rosa los subtokens que están asociados a dicho constraint.

Como se observa en el diagrama, todas las clases que extiendan a *VistaInterface* tienen relación con el Editor y con el Semántico, el primero utiliza las funciones de las vistas para mostrar por pantalla los elementos, y el Semántico informa de las novedades y actualiza las vistas.

9.5.2 Clases Controlador

Los controladores son las clases que reciben información de las vistas y la reenvían al Semántico.

Al igual que las vistas, hay tres clases de Controlador, según si el tipo de datos del sensor es 0D,1D, o 2D. Todas ellas heredan de una clase padre, *ControladorBase*.

- ControladorBase: Incluye todos los procedimientos y funciones de tratamiento de eventos comunes a los controladores.
    - Procedimientos de la clase:
        - clickEnArea: Recibe las coordenadas de un elemento, junto con su longitud y altura, y comprueba si el click del usuario ha sido dentro de dicho elemento.
        - controlEventos: Función *virtual*, es decir, que cada clase que herede del padre tiene que definirla. Cada controlador tiene una lógica de tratamiento de eventos (clicks, teclas de botón) distinto, por lo que debe implementar su propio *handler* de eventos.
        - volverAtras: Función que sirve para volver al menú principal.
        - getTokens: Función que devuelve la lista de Tokens.
        - getTokenByName: Función que recibe un nombre como parámetro y busca el token con dicho nombre.
        - getObjetos: Función que devuelve la lista de objetos.
        - getSubTokens: Función que devuelve la lista de subtokens asociada a un token.
        - borrarConstraintToken: Función que borra un constraint de un token.
        - borrarConstraintSubtoken: Función que borra un constraint de un token
        - addConstraintSubtoken: Función que añade un constraint a un subtoken
        - addConstraintToken: Función que añade un constraint a un token

- Controlador0D: Incluye los atributos y procedimientos para tratar la interacción del usuario con la Vista 0D.

    - Procedimientos de la clase:
        - marcarSubtoken: Añade el subtoken seleccionado al constraint0D.
        - desmarcarSubtoken: Elimina el subtoken seleccionado del constraint0D.
- Controlador1D: Incluye los atributos y procedimientos para tratar la interacción del usuario con la Vista 1D.
    - Procedimientos de la clase:
        - crearRango: Función que crea un nuevo rango para representar la información de un constraint.
        - reordenarRangos: Al subir o bajar la lista de rangos, esta función modifica el índice a partir del cual se muestran los rangos, dependiendo de si el usuario sube o baja la lista.
        - actualizarRangos: Muestra y crea los rangos cuando se importa un fichero TUIML.
        - borrarRango: Borra un rango existente.
        - setValoresRange: Determina los dos valores de un rango, su máximo y su mínimo.
        - getValoresRange: Devuelve el valor máximo y mínimo de un constraint.
        - getPosicionRango: Devuelve la posición de un rango.
- Controlador2D: Incluye los atributos y procedimientos para tratar la interacción del usuario con la Vista 2D.
    - Atributos de la clase:
        - constraintSelecciondo: Variable que apunta al constraint que el usuario ha seleccionado.
        - constraintMoviendo: Variable que apunta al constraint asociado al subtoken que se está moviendo.
        - elegirPuntos: Indica si el usuario está seleccionando los puntos de un constraint.
        - moviendoPunto: Indica si el usuario está moviendo algún punto de algún constraint.
        - indicePuntoMovil: Representa el índice en la lista de puntos de constraintSeleccionado del punto que se está moviendo.
        - subtokenSeleccionado: Variable que indica si el usuario ha seleccionado un subtoken o no.
        - indiceSubtokenSeleccionado: Índice en la lista de subtokens del token que representa el subtoken seleccionado.
    - Procedimientos de la clase:
        - clickEnPunto: Indica si el usuario ha seleccionado un punto de un constraint.
        - dentroConstraint : Indica si el usuario ha seleccionado con el ratón dentro del área de un constraint.
        - constraintMasPequeño: En el caso de que la coordenada selecciona por un usuario se encuentre dentro de varios constraints, devuelve de todos ellos el que tenga un área más pequeña.
        - estaCoordenada: Cuando el usuario quiere añadir un punto, indica si esa coordenada ya pertenece a un punto.

        - mostrarPanelSubtoken: Añade el botón para crear un constraint asociado a un subtoken.
        - puntoALaIzquierda: Devuelve el punto más a la izquierda y arriba de un constraint, para mostrar su información.
        - modificarCoordenadasConstraint: Modifica las coordenadas de un constraint.
        - crearConstraint2D: Crea un constraint2D, y recibe como parámetros todos los campos necesarios para su creación.
        - activarPanelConstraint: Muestra información perteneciente a un constraint seleccionado, como su nombre, los subtokens añadidos…
        - desactivarPanelConstraint: Desactiva la información de un constraint seleccionado.

Como se observa en el diagrama, todos los controladores se comunican con las otras tres clases: los controladores filtran las interacciones y acceden al Semántico para almacenar los cambios, también acceden directamente para actualizar la vista cuando las interacciones no significan cambios en el modelo, como puede ser el scroll de la lista de constraints 1D, y, finalmente, también se comunica directamente con el Editor, aunque en mucha menor medida, como cuando se presiona el botón para volver al menú principal.

9.5.3 Clase Semántico

El semántico es la clase que ya estaba previamente codificada en el toolkit JUGUEMOS, y en el ámbito de este proyecto, actúa tanto como capa datos, como proveedor de algunos servicios, como filtrado de mensajes y exportación/importación de ficheros TUIML.

- Atributos de la clase:
    - listaTokens: Lista con todos los tokens conectados en la aplicación.
    - listaObjetos: Lista de todos los objetos conectados a la aplicación.
    - listaDispositivosConectados: Lista con todos los dispositivos conectados: sensores, host…
- Procedimientos de la clase:
    - recibirMensaje: Función que recibe un mensaje OSC como parámetro, y se encarga de su filtrado.
    - leerJuego: Función que recibe como parámetro un fichero de texto, y se encarga de parsear su estructura TUIML y crear los elementos asociados a dicho fichero.
    - importarTUIML: Recoge los elementos del dominio (Tokens, constraints…) y los vuelca en un fichero con una estructura TUIML correcta.
    - guardarToken: Almacena un nuevo token en la lista de tokens.
    - borrarToken: Borra un token existente de la lista de tokens.
    - addSubtokenAToken: Añade un subtoken a un token existente en la lista de tokens.
    - addConstraintAObjeto: Añade un constraint a un objeto de la lista.
    - addConstraintAToken: Añade un constraint a un token de la lista.
    - addConstraintASubtoken: Añade un constraint a un subtoken asociado a un token.
    - actualizarAnguloConstraint: Modifica el ángulo de un constraint.
    - actualizarAnguloSubtoken: Modifica el ángulo de un subtoken.
    - actualizarCoordenadasSubtoken: Actualiza las coordenadas de un subtoken.

- actualizarCoordenadasConstraint: Actualiza las coordenadas de un constraint.

Como se observa en el diagrama, el Semántico solo recibe consultas de las clases Editor y los Controladores. El Editor relega en el semántico los servicios de filtrado de mensajes y lectura/salvado de ficheros, y los Controladores acceden al Semántico para modificar los tokens u objetos, o para consultarlos.

9.5.4 Clase Editor

El Editor es la clase principal de esta aplicación, ya que se encarga de comunicar a los otros tres tipos de clase, y recibir los eventos de la interacción con el usuario. Actúa como un panel de control para la aplicación, ya que redirige el flujo de datos y eventos hacia las clases adecuadas.

- Atributos de la clase:
  - nombreFichero: Almacena el nombre del fichero donde se guarda la información, o de donde se lee la información.
  - listaImagenes: Lista que contiene las imágenes .JPG que se utilizan en la aplicación.
  - tipoDato: String que indica el tipo de sensor que está siendo interactuado con el usuario. Sirve de condición para guiar al Editor a interactuar con las clases correctas.
  - botonesFichero: Lista de botones que representan todas las opciones de manipulación con ficheros: abrir fichero, borrar contenido del fichero…
  - listaNombres: Lista de campos de texto que representan diversa información: Nombre de los sensores y host conectados, sus IPs…
  - xInicioDisp: Coordenada x en la que empezar a dibujar la información de las listas de dispositivos
  - yInicioDisp: Coordenada x en la que empezar a dibujar la información de las listas de dispositivos
- Procedimientos de la clase:
  - seleccionarFichero: Permite al usuario seleccionar un archivo dentro del sistema de ficheros en el que esté corriendo la aplicación.
  - exportarEnFichero: Delega en el Semántico la exportación de TUIML en el fichero seleccionado.
  - reenviarMensaje: Delega en el Semántico el filtrado de mensajes.
  - controlEventos0D: Ejecuta el controlador de eventos del Controlador 0D.
  - controlEventos1D: Ejecuta el controlador de eventos del Controlador 1D.
  - controlEventos2D: Ejecuta el controlador de eventos del Controlador 2D.
  - controlEventosSensores: Realiza el control de eventos del menú principal.
  - dibujar0D: Ejecuta las funciones para dibujar por pantalla de la Vista 0D.
  - dibujar1D: Ejecuta las funciones para dibujar por pantalla de la Vista 1D.
  - dibujar2D: Ejecuta las funciones para dibujar por pantalla de la Vista 2D.
  - dibujarSensores: Dibuja por pantalla todos los elementos pertenecientes al menú principal.

Como se observa en el diagrama de clases, la clase Editor está conectada con el resto de las clases.

Como se aprecia en el diagrama de secuencia que muestra el flujo de datos (Imagen 26), todas las clases se comunican con el Editor: el Editor utiliza las funciones y componentes gráficos de

las Vistas para representar la información en pantalla, utiliza las funciones de los Controladores para chequear eventos y hacer comprobaciones, y delega en el Semántico servicios y lo utiliza para comprobar los datos ahí almacenados.